(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 576 749 A1**

(12) **EUROPEAN PATENT APPLICATION**

(21) Application number: **92306038.8**

(51) Int. Cl.5: **G06F 9/38**

(22) Date of filing: **30.06.92**

(43) Date of publication of application:
05.01.94 Bulletin 94/01

(84) Designated Contracting States:
AT BE CH DE DK ES FR GB GR IT LI LU MC NL PT SE

(71) Applicant: **PIONEER DIGITAL DESIGN CENTRE LIMITED**
Stanway House,
Almondsbury Business Centre
Woodlands, Almondsbury, Bristol BS12 4QH(GB)

(72) Inventor: **Wise, Adrian Philip**
10 Westbourne Cottages,
Frenchay Common
Frenchay, Bristol BS16 1NA(GB)
Inventor: **Robbins, William Philip**
7 Dunford Road
Bedminster, Bristol BS3 4PN(GB)
Inventor: **Sotheran, Martin William**
The Ridings,
Wick Lane,
Stinchcombe
Dursley, Gloucestershire GL11 6BD(GB)

(74) Representative: **Godsill, John Kenneth et al**
Haseltine Lake & Co.
Hazlitt House
28 Southampton Buildings
Chancery Lane
London WC2A 1AT (GB)

(54) **Data pipeline system and data encoding method.**

(57) A pipeline structure processes data in a series of stages, each of which has a data input latch (LDIN) and passes it on to the next stage in the pipeline via a data output latch (LDOUT). The stages are preferably connected to two non-overlapping clock phases (PH0, PH1). Adjacent stages are also connected via a validation line (IN__VALID, OUT__VALID) and an acceptance line (IN__ACCEPT, OUT__ACCEPT), and in some embodiments also via an extension bit line (IN__EXTN, OUT__EXTN). Input data is transferred from any stage to the following stage once every complete period of the clock signals only if the acceptance signal from that following stage is in the affirmative state. The extension bit line conveys an extension bit that separates data in the data stream into blocks. Decoding circuitry may also be included in any of the stages so that a stage only manipulates the data that is in blocks in which one or more predetermined bit patterns is decoded at the start of the block.

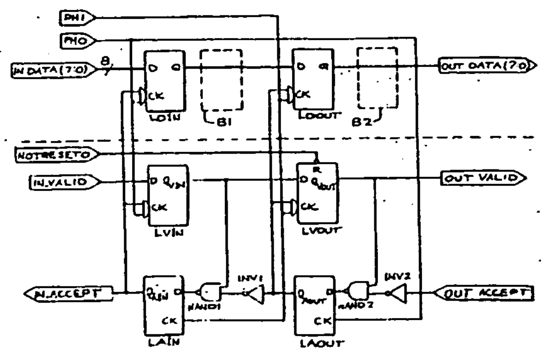


FIG. 4

EP 0 576 749 A1

BEST AVAILABLE COPY

This invention relates to a pipeline of processing stages and storage elements and also relates to a method for encoding data within the pipeline.

Increasing the rate at which digital data can be processed is an important goal in almost every analog and digital design. One known structure for data processing is commonly referred to as a "pipeline". In a typical pipeline structure, data is received at a first processing station or stage and is passed in a predetermined order to subsequent stages in the pipeline. Each pipeline stage may either process the data or simply act as a conduit and pass it on during the next clock cycle of the pipeline.

Existing pipelines, however, are typically "rigid", so that a delay in one stage of the pipeline causes the entire pipeline to stop; the pipeline must wait until the busy stage has completed its task. Since the entire pipeline is stopped, all of the processing stages except the one which is being waited on cannot process data (beyond completing whatever processing they have in hand that does not require data transfers from the preceding stage or to the succeeding stage). In many applications there are several stages in the pipeline that cannot at one time or another proceed and will therefore halt the pipeline. As a result, there is a tendency for these various processing stages to take turns at processing data rather than to work simultaneously on different data in the pipeline. This means that it is possible that much of the potential processing resource that is provided in the pipeline is wasted because each processing stage spends a significant amount of time waiting for other stages. Ideally, all of the processing stages would spend all of their time processing data.

As one example, in many digital television systems, such as the Digital High Definition TV (HDTV) that is expected to replace existing analog technology, the data rate required for transmission is reduced by avoiding the transmission of redundant information. Methods for accomplishing this typically utilize many techniques in concert; it is common for one of these to be the well known technique of entropy coding. In such a system, the television picture is processed into a series of symbols that can be reconstructed (using suitable algorithms embodied in the television receiver) into the picture that will be displayed on the receiver's display device. In order to transmit these symbols from the transmitter to the receiver, each symbol is assigned a unique digital code that identifies it. These digital codes may then be transmitted and received using known techniques for the transmission of digital data.

According to the technique of entropy coding these digital codes are of variable length and are assigned to the symbols they represent with the intention that the most commonly occurring symbols are represented by the shortest codes. In this way, the majority of the symbols that are transmitted are short and require little time for transmission compared to the longer codes that are not often transmitted.

In the television receiver there will be many processing tasks that must be performed one after another to reconstruct the television picture prior to display. For example, one of these tasks will be the decoding of the entropy coded data just described into the stream of symbols that it represents. This data will then usually be written into a digital memory device, possibly after further processing stages. The data that is written into the memory device may be a direct digital representation of the decoded picture or, more likely, a partially decoded representation requiring further processing before the picture is completely decoded. In either case there is a pipeline of processing stages, one of which is the entropy decoder, and the data arriving at the end of the processing pipeline is written into the digital memory.

In such a pipeline there are at least two stages that may hold up the processing of the pipeline. The entropy decoder, for example, will take a variable amount of time to decode the symbols because the digital codes that represent those symbols are of variable length. The digital memory device may also hold up the data because it will not always be possible to write data into the memory as soon as it is received from the pipeline; the memory device will typically be a shared resource with other processes making demands upon it. At the very least there must be something taking data out of the memory again for display or further processing.

In a rigid pipeline scheme, when the entropy decoder is unable to pass decoded symbols onto the succeeding stage in the pipeline, all of the subsequent pipeline stages must wait for it; the digital memory device will then be unable to write any data into the memory because it will not be receiving data from the previous stage. Similarly, when the digital memory device cannot write data that it receives into the memory (because it is busy doing some other function) then the whole pipeline must stop. In this case, the entropy decoder cannot proceed once it has completed decoding the symbol that it has started to decode because it will be unable to pass this symbol on to its succeeding stage in the pipeline.

Another drawback of existing data pipelines is that whenever one stage in the pipeline is delayed, corresponding control signals must be propagated all the way back through earlier stages to the initial pipeline stage in order to prevent further processing until the active stage has completed its task. If it were a stage close to the end of a pipeline that was delayed, this control signal would have to propagate back through most of the earlier pipeline stages. In a pipeline consisting of any significant number of stages it

In addition to clock and data signals (described below), the pipeline includes two transfer control signals -- a "VALID" signal and an "ACCEPT" signal -- to control the transfer of data within the pipeline. The VALID signal, which is illustrated as the upper of two lines connecting neighboring stages, is passed in a forward or downstream direction from each pipeline stage to the nearest neighboring device, which may be another pipeline stage or some other system; for example, the last pipeline stage may pass its data on to subsequent processing circuitry. The ACCEPT signal, which is illustrated as the lower of two lines connecting neighboring stages, passes in the other direction upstream to a preceding device.

Although not shown in Fig. 1, data lines, either single lines or several parallel lines forming a data bus, also lead into and out of each pipeline stage. As is explained and illustrated in greater detail below, data is transferred into, out of, and between the stages of the pipeline over the data lines.

Note that the first pipeline stage may either receive data and control signals from any form of preceding device, (for example, reception circuitry of a digital image transmission system, another pipeline, etc.), or it may itself generate all or part of the data to be processed in the pipeline. Indeed, as is explained below, a "stage" may contain arbitrary processing circuitry, including none at all (for simple passing on of data) and entire systems (for example, another pipeline or even multiple systems or pipelines), and it may generate, change, and delete data as desired.

When a pipeline stage contains valid data that is to be transferred down the pipeline, the VALID signal, which indicates data validity, need not be transferred farther than to the immediately following pipeline stage. A two-wire interface is therefore included between every pair of pipeline stages in the system (and between a preceding device and the first stage, and between a following device and the last stage, if such other devices are included and data is to be transferred between them and the pipeline).

Each of the signals ACCEPT and VALID has a HIGH and a LOW value; these values are abbreviated as "H" and "L", respectively. The most common applications of the pipeline according to the invention will typically be digital; in such digital implementations, the HIGH value may, for example, be a logical "1" and the LOW value may be the logical "0". The invention is not restricted to digital implementations, however, and in analog implementations, the HIGH value may be a voltage or other similar quantity above (or below) a set threshold, with the LOW value being indicated by the corresponding signal being below (or above) the came or some other threshold. For digital applications, the invention may be implemented using any known technology, such as CMOS, bipolar, etc.

It is not necessary according to this invention for a distinct storage device and wires to be provided for storage of VALID signals even in a digital embodiment. All that is requirement is that the notion of "validity" of the data be stored along with the data. By way of example only, in digital television pictures that are represented by digital values as specified in the international standard CCIR 601, certain specific values are not allowed. In this system, eight-bit binary numbers are used to represent samples of the picture and the values zero and 255 may not be used.

If such a picture were to be processed in a pipeline built according to this invention then one of these values (zero, for example) could be used to mean that the data in a specific stage in the pipeline is not valid; thus any non-zero data would be deemed to be valid. In this example, there is no specific distinct latch that can be identified and said to be storing the "validness" of the associated data; nonetheless the validity of the data is stored along with the data.

In Fig. 1, the state of the VALID signal into each stage is indicated as an "H" or an "L" on an upper, right-pointing arrow. The VALID signal from Stage A into Stage B is thus LOW, and the VALID signal from Stage D into Stage E is HIGH. The state of the ACCEPT signal into each stage is indicated as an "H" or an "L" on a lower, left-pointing arrow. The ACCEPT signal from Stage E into Stage D is thus HIGH, whereas the ACCEPT signal from the device connected downstream of the pipeline into Stage F is LOW.

Data is transferred from one stage to another during a cycle (explained below) whenever the ACCEPT signal of the downstream stage into its upstream neighbor is HIGH. If the ACCEPT signal is LOW between two stages, then data is not transferred between these stages.

In Fig. 1, if a box is shaded, the corresponding pipeline stage is assumed, by way of example, to contain valid output data and the VALID signal from that stage into the following stage is HIGH. Fig. 1 illustrates the pipeline when stages B, D, and E contain valid data. Stages A, C, and F do not contain valid data. At the start, the VALID signal into pipeline stage A is HIGH, meaning that the data on the transmission line into the pipeline is valid.

Also at this time, the ACCEPT signal into pipeline stage F is LOW, so that no data, whether valid or not, is transferred out of Stage F. Note that both valid and invalid data is transferred between pipeline stages. Invalid data, which is data not worth saving, may be written over (thereby eliminating it from the pipeline), whereas valid data must not be written over since it is data that must be saved for processing or use in a downstream device (such as a pipeline stage or a device or system connected to the pipeline and receives data from the pipeline).

In the pipeline illustrated in Fig. 1, Stage E contains valid data D1, Stage D contains valid data D2, Stage B contains valid data D3, and a device (not shown) connected to the pipeline upstream contains data D4 that is to be transferred into and processed in the pipeline. Stages B, D, and E, as well as the upstream device, contain valid data and the VALID signal from these stages or devices into the respective following devices is therefore HIGH. The VALID signal from the Stages A, C, and F is, however, LOW since these stages do not contain valid data.

Assume now that the device connected downstream from the pipeline is not ready to accept data from the pipeline. It signals this by setting the corresponding ACCEPT signal LOW into Stage F. Stage F itself, however, does not contain valid data and is therefore able to accept data from the preceding Stage E. The ACCEPT signal from Stage F into Stage E is therefore set HIGH.

Similarly, although Stage E contains valid data, Stage F is ready to accept this data, so that Stage E itself also can accept new data as long as the valid data D1 is first transferred to Stage F. In other words, although Stage F cannot transfer data downstream, all the other stages can do so without any valid data being overwritten and lost. At the end of Cycle 1, data can therefore be "shifted" one step to the right. This condition is shown in Cycle 2.

In the illustrated example, the downstream device is still not ready to accept new data in Cycle 2 so that the ACCEPT signal into Stage F is still LOW. Stage F cannot, therefore, accept new data since doing so would cause valid data D1 to be overwritten and lost. The ACCEPT signal from Stage F into Stage E therefore goes LOW, as does the ACCEPT signal from Stage E into Stage D, since Stage E also contains valid data D2. All of the Stages A-D, however, are able to accept new data (either because they do not contain valid data or because they are able to shift their valid data downstream and accept new) and they signal this condition to their immediately preceding neighbors by setting their corresponding ACCEPT signals HIGH.

The state of the pipelines after Cycle 2 is illustrated in Fig. 1 for the row labelled Cycle 3. By way of example, it is assumed that the downstream device still is not ready to accept new data from Stage F (the ACCEPT signal into Stage F is LOW). Stages E and F therefore still are "blocked", but in Cycle 3, Stage D has received the valid data D3, which overwrote the invalid data that was previously in this stage. Since Stage D cannot pass on data D3 in Cycle 3, it cannot accept new data and therefore sets the ACCEPT signal into Stage C LOW. Stages A-C are, however, ready to accept new data and signal this by setting their corresponding ACCEPT signals HIGH. Note that data D4 has been shifted from Stage A to Stage B.

Assume now that the downstream device becomes ready to accept new data in Cycle 4. It signals this to the pipeline by setting the ACCEPT signal into Stage F HIGH. Although Stages C-F contain valid data, they can now shift the data downstream and are thus able to accept new data. Since each stage is therefore able to shift data one step downstream, they set their respective ACCEPT signals out HIGH.

As long as the ACCEPT signal into the final pipeline stage (in this example, Stage F) is HIGH, the pipeline shown in Fig. 1 thus acts as a rigid pipeline and simply shifts data one step downstream on each cycle. Accordingly, in Cycle 5, data D1, which was contained in Stage F in Cycle 4, is shifted out of the pipeline to the following device, and all other data is shifted one step downstream.

Assume now that the ACCEPT signal into Stage F goes LOW in Cycle 5. Once again, this means that Stages D-F are not able to accept new data, and the ACCEPT signals out of these stages into their immediately preceding neighbors also go LOW. The data D2, D3 and D4 therefore cannot shift downstream, although the data D5 can. The corresponding state of the pipeline after Cycle 5 is thus shown in Fig. 1 as Cycle 6.

The ability of the pipeline according to the preferred embodiments of the invention to "fill up" empty processing stages is highly advantageous since the processing stages in the pipeline thereby become decoupled from one another. In other words, even though a pipeline stage may not be ready to accept data, the entire pipeline does not have to stop and wait for the delayed stage. Rather, when one stage is not able to accept valid data it simply forms a temporary "wall" in the pipeline; stages downstream of the "wall" can continue to advance valid data even to the circuitry connected to the pipeline, and stages to the left of the "wall" stage can still accept and transfer valid data downstream. Even when several pipeline stages temporarily cannot accept new data, other stages can continue to operate normally. In particular, the pipeline can continue to accept data into its initial stage A as long as stage A does not already contain valid data that cannot be advanced due to the next stage not being ready to accept new data. As this example illustrates, data can be transferred into the pipeline and between stages even when one or more processing stages is blocked.

In the embodiment of the invention shown in Fig. 1, it is assumed that the various pipeline stages do not store the ACCEPT signals they receive from their immediately following neighbors; instead, whenever the ACCEPT signal into a downstream stage goes LOW, this LOW signal is propagated upstream as far as to the nearest pipeline stage that does not contain valid data. For example, referring to Fig. 1, it was assumed that the ACCEPT signal into Stage F goes LOW in Cycle 1. In Cycle 2, the LOW signal propagates from Stage F back to Stage D.

In Cycle 3, when the data D3 is latched into Stage D, the ACCEPT signal propagates upstream four stages to Stage C. When the ACCEPT signal into Stage F goes HIGH in Cycle 4, it must propagate upstream all the way to Stage C; in other words, the change in the ACCEPT signal must propagate back four stages. It is not necessary, however, in the embodiment illustrated in Fig. 1 for the ACCEPT signal to propagate all the way back to the beginning of the pipeline if there is some intermediate stage that is able to accept new data.

In the embodiment illustrated in Fig. 1, each pipeline stage will still need separate input and output data latches in order to allow data to be transferred between stages without unintended overwriting. Also, although the pipeline illustrated in Fig. 1 is able to "compress" when downstream pipeline stages are blocked (cannot pass on the data they contain), the pipeline does not "expand" to create stages that do not contain valid data between stages that contain valid data. Rather, the ability to compress depends on there being cycles during which no valid data is presented to the first pipeline stage.

In Cycle 4, for example, if the ACCEPT signal into Stage F remained LOW and valid data filled pipeline stages A and B, as long as valid data continued to be presented to Stage A the pipeline would not be able to compress any further and valid input data could be lost. Nonetheless, the pipeline illustrated in Fig. 1 reduces the risk of data loss since it is able to compress as long as there is some pipeline stage that does not contain valid data.

Fig. 2 illustrates an embodiment of the invention that can both compress and expand in a logical manner and which includes circuitry that limits propagation of the ACCEPT signal to the nearest preceding stage. Although the circuitry for implementing this embodiment is explained and illustrated in greater detail below, Fig. 2 serves to illustrate the principle by which it operates.

For ease of comparison only, the input data and ACCEPT signals into the pipeline embodiment shown in Fig. 2 are the same as in the pipeline embodiment shown in Fig. 1. Accordingly, stages E, D, and B contain valid data D1, D2, and D3, respectively; the ACCEPT signal into Stage F is LOW; and data D4 is presented to the beginning pipeline Stage A. In Fig. 2, three lines are shown connecting each neighboring pair of pipeline stages. The uppermost line, which may be a bus, is a data line; the middle line is the line over which the VALID signal is transferred; and the bottom line is the line over which the ACCEPT signal is transferred. Also as before, the ACCEPT signal into Stage F remains LOW except in Cycle 4. Furthermore, additional data D5 is presented to the pipeline in Cycle 4.

In Fig. 2, each pipeline stage is represented as a block divided into two halves to illustrate that each stage in this embodiment of the pipeline includes primary and secondary data storage elements. In Fig. 2, the primary data storage is shown as the right half of each stage, although this is for purposes of illustration only.

As Fig. 2 illustrates, as long as the ACCEPT signal into a stage is HIGH, data is transferred from the primary storage elements of the stage to the secondary storage elements of the following stage during any given cycle. Accordingly, although the ACCEPT signal into Stage F is LOW, the ACCEPT signal into all other stages is HIGH so that the data D1, D2, and D3 is shifted forward one stage in Cycle 2 and the data D4 is shifted into the first Stage A.

Up to this point, the pipeline embodiment shown in Fig. 2 therefore acts in a manner similar to the pipeline embodiment shown in Fig. 1. The ACCEPT signal from Stage F into Stage E, however, is HIGH even though the ACCEPT signal into Stage F is LOW. As is explained below, because of the secondary storage elements, it is not necessary for the LOW ACCEPT signal to propagate upstream beyond Stage F. By leaving the ACCEPT signal into Stage E HIGH, moreover, Stage F signals that it is ready to accept new data. Since Stage F is not able to transfer the data D1 in its primary storage elements downstream (the ACCEPT signal into Stage F is LOW), in Cycle 3, Stage E therefore transfers the data D2 into the secondary storage elements of Stage F. Since both the primary and the secondary storage elements of Stage F then contain valid data that cannot be passed on, the ACCEPT signal from Stage F into Stage E is set LOW -- this represents a propagation of the LOW ACCEPT signal back only one stage relative to Cycle 2, whereas this ACCEPT signal had to be propagated back all the way to Stage C in the embodiment shown in Fig. 1.

Since Stages A-E are able to pass on their data, the ACCEPT signals from the stages into their immediately preceding neighbors are set HIGH. Consequently, the data D3 and D4 are shifted one stage to the right so that, in Cycle 4, they are loaded into the primary data storage elements of Stage E and Stage C, respectively. Although Stage E now contains valid data D3 in its primary storage elements, its secondary storage elements can still be used to store other data without risk of overwriting any valid data.

Assume now as before that the ACCEPT signal into Stage F becomes HIGH in Cycle 4. This indicates that the downstream device to which the pipeline passes data is ready to accept data from the pipeline. Stage F, however, has set its ACCEPT signal LOW and thus indicates to Stage E that Stage F is not prepared to accept new data. Observe that the ACCEPT signals for each cycle indicate what will "happen" in the next cycle, that is, whether data will be passed on (ACCEPT HIGH) or whether data must remain in place (ACCEPT LOW). From Cycle 4 to Cycle 5 the data D1 is therefore passed from Stage F to the following device, the data D2 is shifted from secondary to primary storage in Stage F, but the data D3 in Stage E is not transferred to Stage F. The data D4 and D5 can be transferred into the following pipeline stages as normal since the following stages have their ACCEPT signals HIGH.

Comparing the state of the pipeline in Cycle 4 and Cycle 5, it can be seen that the provision of secondary storage elements enables the pipeline embodiment shown in Fig. 2 to expand, that is, to free up data storage elements into which valid data can be advanced. For example, in Cycle 4, the data blocks D1, D2, and D3 form a "solid wall" since their data cannot be transferred until the ACCEPT signal into Stage F goes HIGH. Once this signal does become HIGH, however, data D1 is shifted out of the pipeline, data D2 is shifted into the primary storage elements of Stage F, and the secondary storage elements of Stage F become free to accept new data if the following device is not able to receive the data D2 and the pipeline must once again "compress"; this is shown in Cycle 6, for which the data D3 has been shifted into the secondary storage elements of Stage F and the data D4 has been passed on from Stage D to Stage E as normal.

Figs. 3a(1), 3a(2), 3b(1), and 3b(2) (which are referred to collectively as Fig. 3) illustrate generally a preferred embodiment of the pipeline according to the invention. This preferred embodiment implements the structure shown in Fig. 2 using a two-phase, non-overlapping clock with phases $\phi 0$ and $\phi 1$. Although a two-phase clock is preferred, it is also possible to drive the various embodiments of the invention using a clock with more than two phases.

In Fig. 3, each pipeline stage is represented as two separate boxes illustrating the primary and secondary storage elements. Also, although the VALID signal and the data lines connect the various pipeline stages as before, for ease of illustration, only the ACCEPT signal is shown in Fig. 3. A change of state during a clock phase of certain of the ACCEPT signals is indicated in Fig. 3 using an upward-pointing arrow for changes from LOW to HIGH, and a downward-pointing arrow for changes from HIGH to LOW. Transfer of data from one storage element to another is indicated by a large open arrow. It is assumed that the VALID signal out of the primary or secondary storage elements of any given stage is HIGH whenever the storage elements contain valid data.

In Fig. 3, each cycle is shown as consisting of a full period of the non-overlapping clock phases $\phi 0$ and $\phi 1$. As is explained in greater detail below, data is transferred from the secondary storage elements (shown as the left box in each stage) to the primary storage elements (shown as the right box in each stage) during clock cycle $\phi 1$, whereas data is transferred from the primary storage elements of one stage to the secondary storage elements of the following stage during the clock cycle $\phi 0$. Fig. 3 also illustrates that the primary and secondary storage elements in each stage are also connected via an internal acceptance line to pass an ACCEPT signal in the same manner that the ACCEPT signal is passed from stage to stage.

Fig. 3 shows the $\phi 1$ phase of Cycle 1, in which data D1, D2, and D3, which were previously shifted into the secondary storage elements of Stages E, D, and B, respectively, are shifted into the primary storage elements of the respective stage. During the $\phi 1$ phase of Cycle 1, the pipeline therefore assumes the same configuration as is shown as Cycle 1 in Fig. 2. As before, the ACCEPT signal into Stage F is assumed to be LOW. As Fig. 3 illustrates, however, this means that the ACCEPT signal into the primary storage element of Stage F is LOW, but since this storage element does not contain valid data, it sets the ACCEPT signal into its secondary storage element HIGH.

The ACCEPT signal from the secondary storage elements of Stage F into the primary storage elements of Stage E is also set HIGH since the secondary storage elements of Stage F do not contain valid data. As before, since the primary storage elements of Stage F are able to accept data, data in all the upstream primary and secondary storage elements can be shifted downstream without any valid data being overwritten. The shift of data from one stage to the next takes place during the next $\phi 0$ phase in Cycle 2. For example, the valid data D1 contained in the primary storage element of Stage E is shifted into the secondary storage element of Stage F, the data D4 is shifted into the pipeline, that is, into the secondary storage element of Stage A, etc.

The primary storage element of Stage F still does not contain valid data during the $\phi 0$ phase in Cycle 2, so that the ACCEPT signal from the primary storage elements into the secondary storage elements of Stage F remains HIGH. During the $\phi 1$ phase in Cycle 2, data can therefore be shifted yet another step to the right, which is from the secondary to the primary storage elements within each stage.

Once valid data is loaded into the primary storage elements of Stage F, however, if the ACCEPT into Stage F from the downstream device is still LOW, it is not possible to shift data out of the secondary storage element of Stage F without overwriting and destroying the valid data D1. The ACCEPT signal from the primary storage elements into the secondary storage elements of Stage F therefore goes LOW. Data D2, however, can still be shifted into the secondary storage of Stage F since it did not contain valid data and its ACCEPT signal out was HIGH.

During the $\phi1$ phase of Cycle 3, it not possible to shift data D2 into the primary storage elements of Stage F, although data can be shifted within all the previous stages. Once valid data is loaded into the secondary storage elements of Stage F, however, Stage F is not able to pass on this data, which it signals by setting its ACCEPT signal out LOW. Assuming that the ACCEPT signal into Stage F remains LOW, data upstream of Stage F can continue to be shifted between stages and within stages on the respective clock phases until the next valid data block D3 reaches the primary storage elements of Stage E. As illustrated, this condition is reached during the $\phi1$ phase of Cycle 4.

During the $\phi0$ phase of Cycle 5, data D3 has been loaded into the primary storage element of Stage E. Since this data cannot be shifted further, the ACCEPT signal out of the primary storage elements of Stage E is set LOW. Upstream data can be shifted as normal.

Assume now as in Cycle 5 of Fig. 2 that the device connected downstream of the pipeline is able to accept pipeline data. It signals this by setting the ACCEPT signal into pipeline Stage F HIGH during the $\phi1$ phase of cycle 4. The primary storage elements of Stage F can now shift data to the right and they are also able to accept new data: the data D1 was shifted out during the $\phi1$ phase of Cycle 5 so that the primary storage elements of Stage F no longer contain data that must be saved. During the $\phi1$ phase of Cycle 5, the data D2 is therefore shifted within Stage F from the secondary storage elements to the primary storage elements. The secondary storage elements of Stage F then are also able to accept new data and signal this by setting the ACCEPT signal into the primary storage elements of Stage E HIGH. During transfer of data within a stage, that is from its secondary to its primary storage elements, both sets of storage elements will contain the same data, but the data in the secondary storage elements can be overwritten with no data loss since this data will also be in the primary storage elements. The same holds for data transfer from the primary storage elements of one stage into the secondary storage elements of a subsequent stage.

Assume now that the ACCEPT signal into the primary storage elements of Stage F goes LOW during the $\phi1$ phase in Cycle 5. This means that Stage F is not able to transfer the data D2 out of the pipeline. Stage F consequently sets the ACCEPT signal from its primary to its secondary storage elements LOW to prevent overwriting of the valid data D2. The data D2 stored in the secondary storage elements of Stage F, however, can be overwritten without loss, and the data D3 is therefore transferred into the secondary storage elements of Stage F during the $\phi0$ phase of Cycle 6. Data D4 and D5 can be shifted downstream as normal. Once valid data D3 is stored in Stage F along with data D2, as long as the ACCEPT signal into the primary storage elements of Stage F is LOW, neither can the secondary storage elements accept new data, and it signals this by setting the ACCEPT signal into Stage E LOW.

When the ACCEPT signal into the pipeline from the downstream device changes from LOW to HIGH or vice versa, this change does not have to propagate upstream within the pipeline farther than to the immediately preceding storage elements (within the same stage or within the preceding pipeline stage); instead, this change propagates upstream within the pipeline one storage element block per clock phase.

As this example illustrates, the concept of a "stage" in the pipeline structure illustrated in Fig. 3 is to some extent a matter of perception. Since data is transferred within a stage (from the secondary to the primary storage elements) as it is between stages (from the primary storage elements of the upstream stage into the secondary storage elements of the neighboring downstream stage), one could just as well consider a stage to consist of "primary" storage elements followed by "secondary storage element" instead of as illustrated in Fig. 3. The concept of "primary" and "secondary" storage elements is also therefore mostly a question of labelling. In Fig. 3, the "primary" storage elements can also be referred to as "output" storage elements, since they are the elements from which data is transferred out of a stage into a following stage or device, and the "secondary" storage elements could be "input" storage elements for the same stage.

In the explanation of the embodiments of the invention shown in Figs. 1-3, only the transfer of data under the control of the ACCEPT and VALID signals has been mentioned. It is to be understood that each pipeline stage may also process the data it has received arbitrarily before passing it between its internal storage elements or before passing it to the following pipeline stage. Referring once again to Fig. 3, a pipeline stage can therefore be defined as the portion of the pipeline that contains input and output storage elements and that arbitrarily processes data stored in its storage elements.

The "device" downstream from the pipeline Stage F, moreover, need not be some other type of hardware structure, but rather could be another section of the same or even of another pipeline. As is illustrated below, a pipeline stage can set its ACCEPT signal LOW not only when all of the downstream storage elements are filled with valid data, but also when a stage requires more than one clock phase to finish processing its data, or when it creates valid data in one or both of its storage elements. It is also not necessary for a stage simply to pass on the ACCEPT signal based on whether or not the immediately downstream storage elements contains valid data that cannot be passed on. Instead, the ACCEPT signal itself may also be altered within the stage or even by circuitry external to the stage in order to control the passage of data between adjacent storage elements; the VALID signal may also be processed in an analogous manner.

A great advantage of the two-wire interface (one wire for each of the VALID and ACCEPT signals) is its ability to control the pipeline without the control signals needing to propagate back up the pipeline all the way to its beginning stage. In the embodiment shown in Fig. 1, Cycle 3, for example, although stage F "tells" stage E that it cannot accept data, and stage E tells stage D, and stage D tells stage C; indeed, if there had been more stages containing valid data then this signal would have propagated back even farther along the pipeline. In the embodiment shown in Fig. 3, Cycle 3, the LOW ACCEPT signal is not propagated any father upstream than to Stage E, and even then only to its primary storage elements.

As is described below, this embodiment is able to achieve this flexibility without adding significantly to the silicon area that is required to implement the design. Typically, each latch in the pipeline that is used for data storage may require as little as only one extra transistor (which lays out very efficiently in silicon). In addition, two extra latches and a small number of gates are preferably added to process the ACCEPT and VALID signals that are associated with the data latches in each half-stage.

Fig. 4 illustrates a hardware structure that implements a stage as shown in Fig. 3.

By way of example only, it is assumed that eight-bit data is to be transferred (with or without further manipulation in optional combinatorial logic circuits) in parallel through the pipeline. The two-wire interface according to this embodiment is, however, suitable for use with any data bus width, and the data bus width may even change from one stage to the next if a particular application so requires. The interface according to this embodiment can also be used to process analog signals.

Although other conventional timing arrangements may be used, the interface is preferably controlled by a two-phase, non-overlapping clock. In Figs. 4-9, these clock phase signals are referred to as PH0 and PH1. In Fig. 4, a line is shown for each clock phase signal.

Input data enters a pipeline stage over a multi-bit data bus IN_DATA and is transferred to a following pipeline stage or to subsequent receiving circuitry over an output data bus OUT_DATA. The input data is first loaded in a manner described below into a series of input latches (one for each input data signal) collectively referred to as LDIN, which constitute the secondary storage elements described above.

In the illustrated example of this embodiment, it is assumed that the Q outputs of all latches follow their D inputs, that is, they are "loaded", when the clock input is HIGH, that is, at a logic "1" level. Additionally, the Q outputs hold their last values, or, in other words, the Q outputs are "latched", on the falling edge of their respective clock signals. Each latch has for its clock either one of two non-overlapping clock signals PH0 or PH1 (shown in Fig. 5), or the logical AND combination of one of these clock signals PH0, PH1 and one logic signal. The invention works just as well, however, by providing latches that latch on the rising edges of the clock signals, or any other known latching arrangement, as long as conventional methods are applied to ensure proper timing of the latching operations.

The output data from the input data latch LDIN passes via an arbitrary and optional combinatorial logic circuit B1, which may be provided to convert output data from input latch LDIN into intermediate data, which is then later loaded in an output data latch LDOUT, which comprises the primary storage elements described above. The output from the output data latch LDOUT may similarly pass through an arbitrary and optional combinatorial logic circuit B2 before being passed onward as OUT_DATA to the next device downstream, which may be another pipeline stage or any other device connected to the pipeline.

Each stage of the pipeline also includes a validation input latch LVIN, a validation output latch LVOUT, an acceptance input latch LAIN, and an acceptance output latch LAOUT. Each of these four latches is preferably a simple, single-stage latch. The outputs from latches LVIN, LVOUT, LAIN and LAOUT are, respectively, QVIN, QVOUT, QAIN, QAOUT. The output signal QVIN from the validation input latch is connected either directly as an input to the validation output latch LVOUT, or via intermediate logic devices or circuits that may alter the signal.

Similarly, the output validation signal QVOUT of a given stage may be connected either directly to the input of the validation input latch QVIN of the following stage, or via intermediate devices or logic circuits, which may even alter the validation signal. This output QVIN is also connected to a logic gate (to be

As an example of the operation of a pipeline augmented by the two-wire validation and acceptance circuitry according to the embodiment, assume that no valid data is initially presented at the input to the circuit, either from a preceding pipeline stage, or from a transmission device. In other words, assume that the validation input signal IN__VALID the illustrated stage has not gone to a "1" since the system was most recently reset. Assume also that several clock cycles have taken place since the system was last reset so that the circuitry has reached a steady-state condition. The validation input signal QVIN from the validation latch LVIN is therefore loaded as a "0" during the next positive period of the clock PH0. The input to the acceptance input latch LAIN (via the gate NAND1 or another equivalent gate), is therefore loaded as a "1" during the next positive period of the clock signal PH1. In other words, since the data in the data input latch LDIN is not valid, the stage signals that it is ready to accept input data (since it does not hold any data worth saving).

In this example, note that the signal IN__ACCEPT is used to enable the data and validation latches LDIN and LVIN. Since the signal IN__ACCEPT at this time is a "1", these latches effectively work as conventional transparent latches so that whatever data is on the IN__DATA bus is simply loaded into the data latch LDIN as soon as the clock signal PH0 goes to a "1". Of course, this invalid data will also be loaded into the next data latch LDOUT of the following pipeline stage as long as the output QAOUT from its acceptance latch is a "1".

In other words, as long as a data latch does not contain valid data, it accepts or "loads" any data presented to it during the next positive period of its respective clock signal. On the other hand, such invalid data is not loaded in any stage for which the acceptance signal from its corresponding acceptance latch is low (that is, a "0"). Furthermore, the output signal from a validation latch (which forms the validation input signal to the subsequent validation latch) remains a "0" as long as the corresponding IN__VALID (or QVIN) signal to the validation latch is low.

When the input data to a data latch is valid, the validation signal IN__VALID indicates this by rising to a "1". The output of the corresponding validation latch then rises to a "1" on the next rising edge of its respective clock phase signal. For example, the validation input signal QVIN of latch LVIN rises to a "1" when its corresponding IN__VALID signal goes high (that is, rises to a "1") on the next rising edge of the clock phase signal PH0.

Assume now, instead, that the data input latch LDIN contains valid data. If the data output latch LDOUT is ready to accept new data, its acceptance signal QAOUT will be a "1". In this case, during the next positive period of the clock signal PH1, the data latch LDOUT and validation latch LVOUT will be enabled, and the data latch LDOUT will load the data present at its input. This will occur before the next rising edge of the other clock signal PH0, since the clock signals are non-overlapping. At the next rising edge of PH0, the preceding data latch (LDIN) will therefore not latch in new input data from the preceding stage until the data output latch LDOUT has safely latched the data transferred from the latch LDIN.

The same sequence is followed by every adjacent pair of data latches (within a stage or between adjacent stages) that are able to accept data, since they will be operating based on alternate phases of the clock. Any data latch that is not ready to accept new data because it contains valid data that cannot yet be passed on will have an output acceptance signal (the QA output from its acceptance latch LA) that is LOW, and its data latch LDIN or LDOUT will not be loaded. In other words, as long as the acceptance signal (the output from the acceptance latch) of a given stage or side (input or output) of a stage is LOW, its corresponding data latch will not be loaded.

Fig. 4 also shows a reset feature included in the preferred embodiment of the invention. In the illustrated example, a reset signal NOTRESET0 is connected to an inverting reset input R (inversion is hereby indicated by a small circle, as is conventional) of the validation output latch LVOUT. As is well known, this means that the validation latch LVOUT will be forced to output a "0" whenever the reset signal NOTRESET0 becomes a "0". One advantage of resetting the latch when the reset signal goes low (becomes a "0") is that a break in transmission will reset the latches; they will then be in their "null" or reset state whenever a valid transmission begins and the reset signal goes HIGH. The reset signal NOTRESET0 therefore operates as a digital "ON/OFF" switch, such that it must be at a HIGH value in order to activate the pipeline.

Note that it is not necessary to reset all of the latches that hold valid data in the pipeline. In Fig. 4, the validation input latch LVIN is not directly reset by the reset signal NOTRESET0, but rather is reset indirectly. Assume that the reset signal NOTRESET0 drops to a "0". The validation output signal QVOUT then also drops to a "0", regardless of its previous state, whereupon the input to the acceptance output latch LAOUT (via the gate NAND1) goes HIGH. The acceptance output signal QAOUT then also rises to a "1". This QAOUT value of "1" is then transferred as a "1" to the input of the acceptance input latch LAIN regardless of the state of the validation input signal QVIN. The acceptance input signal QAIN then rises to a

"1" at the next rising edge of the clock signal PH1. Assuming that the validation signal IN__VALID has been correctly reset to a "0", then upon the subsequent rising edge of the clock signal PH0, the output from the validation latch LVIN will become a "0", as it would have done if it had been reset directly.

As this example illustrates, it is only necessary to reset the validation latch in only one side of each stage (including the final stage) in order to reset all validation latches. In fact, in many applications, it will not even be necessary to reset every other validation latch: If the reset signal NOTRESET0 can be guaranteed to be low during more than one complete cycle of both phases PH0, PH1 of the clock, then the "automatic reset" (a backwards propagation of the reset signal) will occur for validation latches in preceding pipeline stages. Indeed, as long as the reset signal is held low for at least as many as full cycles of both phases of the clock as there are pipeline stages, it would only be necessary to reset directly the validation output latch in the final pipeline stage.

Figs. 5a and 5b (referred to collectively as Fig. 5) illustrate a timing diagram showing the relationship between the non-overlapping clock signals PH0, PH1, the effect of the reset signal, and the holding and transfer of data for the different permutations of validation and acceptance signals into and between the two illustrated sides of a pipeline stage configured as in the embodiment shown in Fig. 4. In the example illustrated in the timing diagram of Fig. 5, it has been assumed that the outputs from the data latches LDIN, LDOUT are passed without further manipulation by intervening logic blocks B1, B2. This is by way of example only, and it is to be understood that any combinatorial logic structures may be included between the data latches of consecutive pipeline stages, or between the input and output sides of a single pipeline stage. The actual illustrated values for the input data (for example the HEX data words "aa" or "04") are also merely illustrative. As is mentioned above, the input data bus may have any width (and may even be analog), as long as the data latches or other storage devices are able to accommodate and latch or store each bit or value of the input word.

Preferred Data Structure - "tokens"

In the simple application shown in Fig. 4, each stage processes all input data, since there is no control circuitry that excludes any stage from allowing input data to pass through its combinatorial logic block B1, B2, etc. To provide greater flexibility, this embodiment includes a data structure in which "tokens" are used to distribute data around the system. Each token consists of a series of binary bits separated into one or more blocks of token words and into one of three types: address bits (A), data bits (D), and an extension bit (E). Assuming by way of example only that data is transferred as words over an 8-bit bus with a 1-bit extension bit line, an example of a four-word token is, in order of transmission:

First word:	E	A	A	A	D	D	D	D	D
Second word:	E	D	D	D	D	D	D	D	D
Third word:	E	D	D	D	D	D	D	D	D
Fourth word:	E	D	D	D	D	D	D	D	D

Note that the extension bit E is used as an addition (preferably) to each data word and that the address field can be of variable length and is preferably transmitted just after the extension bit of the first word.

Tokens thus consist of one or more words of (binary) digital data. Each of these words is transferred in sequence and preferably in parallel, although this method of transfer is not necessary: serial data transfer is also possible using known techniques.

As the example illustrates, each token has, preferably at the start, an address field (the string of A-bits) that identifies the type of data that is contained in the token. In many applications, a single word or part of a word is enough to transfer the entire address field, but this is not necessary according to this embodiment as long as logic circuitry is included in the corresponding pipeline stages that is able to store some representation of partial address fields long enough for the stages to receive and decode the entire address field.

Note that no dedicated wires or registers are required to transmit the address field -- it is transmitted using the data bits. As is explained below, a pipeline stage will not be slowed down if it is not intended to be activated by the particular address field; the stage will be able to pass along the token without delay.

The remainder of the data in the token after the address field is not constrained by the use of tokens. These D-data bits may take on any values and the meaning attached to these bits is of no importance here. The number of data bits D appended after the address field can be as long or as short as required, and the number of data words in different tokens may vary greatly. The address field and extension bit are used to convey control signals to the pipeline stages. Because the number of words in the data field (the string of D bits) can be arbitrary, as can be the information conveyed in the data field. The explanation below is therefore directed to the use of the address and extension bits.

Tokens are a particular useful data structure when a number of blocks of circuitry are connected together in a relatively simple configuration. The simplest configuration is a pipeline of processing steps, for example, the one shown in Fig. 1; the use of tokens is not, however, restricted to use on a pipeline structure.

Assume once again that each box represents a complete pipeline stage. In the pipeline of Fig. 1, data flows from left to right in the diagram. Data enters the machine and passes into processing Stage A. This may modify the data and it then passes the data, modified or unmodified, to Stage B. The modification may be arbitrarily complicated and in general there will not be the same number of data items flowing into any stage as flow out. Stage B modifies the data again and passes it onto Stage C, etc. In a scheme such as this it is impossible for data to flow in the opposite direction, so that, for example, Stage C cannot pass data to Stage A for example. This restriction is often perfectly acceptable.

On the other hand, it is very desirable for Stage A to be able to communicate information to Stage C even though there is no direct connection between the two blocks, but only via Stage B. One advantage of the tokens is their ability to achieve this kind of communication. Any processing stage that does not recognize a token passes it on unaltered to the next block.

According to this example, an extension bit is transmitted along with the address and data fields in each token so that a processing stage can pass on a token (which can be of arbitrary length) without having to decode its address at all. According to this example, any token in which the extension bit is HIGH (a "1") is followed by a subsequent word which is part of the same token. This word also has an extension bit, which indicates whether there is a further token word in the token. When a stage encounters a token word whose extension bit is LOW (a "0") then it is known to be the last word of the token. The next word is then assumed to be the first word of a new token.

Note that although the simple pipeline of processing stages is particularly useful, tokens may be applied to more complicated configurations of processing elements. An example of such a more complicated processing element is described below.

It is not necessary according to this embodiment to use the state of the extension bit to signal the last word of a given token by giving it an extension bit set to "0". One alternative to the preferred scheme is to move the extension bit so that it indicates the first word of a token instead of the last, with appropriate changes in the decoding hardware.

The advantage of using the extension bit to signal the last word in a token rather than the first is that it is often useful to modify the behavior of a block of circuitry depending upon whether or not a token has extension bits. An example of this would a token that activates a stage that processes video quantization values stored in a quantization table (typically a memory device), for example, a table containing 64 eight-bit arbitrary binary integers.

In order to load a new quantization table into the quantizer stage of the pipeline, a "QUANT_TABLE" token is sent to the quantizer. In such a case the token would for example consist of 65 token words, the first of which would contain the code "QUANT_TABLE", followed by 64 words, which are the integers of the quantization table.

When encoding video data it is occasionally necessary to transmit such a quantization table. In order to do this, a QUANT_TABLE token with no extension words can be sent to the quantizer stage. On seeing this token, and noting that the extension bit of its first word is LOW, the quantizer stage can read out its quantization table and construct a QUANT_TABLE token which includes the 64 quantization table values. The extension bit of the first word (which was LOW) is changed so that it is HIGH and the token then continues, with HIGH extension bits, until the new end of the token, indicated by a LOW extension bit on the sixty fourth quantization table value. This proceeds in the normal way through the system and is encoded into the bit stream.

Continuing with the example, the quantizer may either load a new quantization table into its own memory device or read out its table depending on whether the first word of the QUANT_TABLE token has its extension bit set or not.

The choice of whether to use the extension bit to signal the first or last token word in a token will therefore depend on the system in which the pipeline will be used. Both alternatives are possible according to the invention.

Another alternative to the preferred extension bit scheme is be to include a length count at the start of the token. Such an arrangement may, for example, be efficient if a token is very long. For example, assume that a typical token in a given application is 1000 words long. Using the illustrated extension bit scheme (with the bit attached to each token word), the token would require 1000 additional bits to contain all the extension bits. Only ten bits would be required, however, to encode the token length in binary form.

Although there are therefore uses for long tokens, experience has shown that there are very many uses for short tokens. Here the preferred extension bit scheme is advantageous. If a token is only one word long then only one bit is required to signal this but a counting scheme would typically require the same ten bits as before.

Disadvantages of a length count scheme include the following: 1) it is inefficient for short tokens; 2) it places a maximum length restriction on a token (with only ten bits, no more than 1023 words can be counted); 3) the length of a token must be known in advance of generating the count (which is presumably at the start of the token); 4) every block of circuitry that deals with tokens would need to be provided with hardware to count words; and 5) if the count should get corrupted (due to a data transmission error) then it is not clear that recovery can ever be achieved.

The advantages of the extension bit scheme according to this embodiment include: 1) no pipeline stage needs to include a block of circuitry that decodes every token --unrecognized tokens can be passed on correctly by considering only the extension bit; 2) the coding of the extension bit is identical for all tokens; 3) there is no limit placed on the length of a token; 4) the scheme is efficient (in terms of overhead to represent the length of the token) for short tokens; and 5) error recovery is naturally achieved: if an extension bit is corrupted then one random token will be generated (for an extension bit corrupted from "1" to "0") or a token will be lost (extension bit corrupted "0" to "1"); furthermore, the problem is localized to the tokens concerned -- after this, correct operation is resumed automatically.

The length of the address field may be varied. This is highly advantageous since it allows the most common tokens to be squeezed into the minimum number of words. This in turn is of great importance in video data pipeline systems since it ensures that all processing stages can be continuously running at full bandwidth.

According to this embodiment, in order to allow variable length address fields the addresses are chosen so that a short address followed by random data can never be confused with a longer address. The preferred technique for encoding the address field (which also serves as the "code" for activating an intended pipeline stage) is the well-known technique first described by Huffman, hence the common name "Huffman Code".

Although Huffman encoding is well understood in the field of digital design, the following example provides a general background:

Huffman codes consist of words made up of a string of symbols (in the context of digital systems such as this invention, the symbols are usually binary digits). The code words may have variable length and the special property of Huffman code words is that a code word is chosen so that none of the longer code words starts with the symbols that form a shorter code word. According to the invention, token address fields are preferably (although not necessarily) chosen using known Huffman encoding techniques.

Also according to this embodiment, the address field preferably starts in the most significant bit (MSB) of the first word token. (Note that the designation of the MSB is arbitrary and that this scheme can be modified to accommodate various designations of the MSB.) The address field continues through contiguous bits of lesser significance. If, in a given application, a token address requires more than one token word, the least significant bit in any given word the address field will continue in the most significant bit of the next word. The minimum length of the address field is one bit.

Any of several known hardware structures can be used to generate the tokens used in this example . One such structure is a microprogrammed state machine, but known microprocessors or other devices may also be used.

The principle advantage of the token scheme according to this example is its adaptability to unanticipated needs. For example, if a new token is introduced then it is most likely that this will affect only a small number of pipeline stages. The most likely case is when only two stages or blocks of circuitry are affected -- the one block that generates the tokens in the first place and the block or stage that has been newly designed or modified to deal with this new token. Note that it is not necessary to modify any other pipeline stages; rather, these will be able to deal with the new token without modification to their designs because they will not recognize it and will pass that token on unmodified.

This ability of this embodiment to leave substantial existing designed devices unaffected has clear advantages. It may be possible to leave some semiconductor chips in a chip set completely unaffected by a design improvement in some other chips in the set. This is advantageous both from the perspective of a customer and from that of a chip manufacturer. Even if modifications mean that all chips are affected by the design change (a situation that becomes increasingly likely as levels of integration progress so that the number of chips in a system drops) there will still be the considerable advantage of better time-to-market than can be achieved, since the same design can be reused.

In particular, note the situation that occurs when it becomes necessary to extend the token set to include twoword addresses. Even in this case it is still not necessary to modify an existing design. Token decoders in the pipeline stages will attempt to decode the first word of such a token and will conclude that it does not recognize the token. It will then pass on the token unmodified using the extension bit to correctly do this. It will not attempt to decode the second word of the token (even though this contains address bits) because it will "assume" that the second word is part of the data field of a token that it does not recognize.

In many cases, a pipeline stage or a connected block of circuitry will modify a token. This usually, but not necessarily, takes the form of modifying the data field of a token. In addition, it is common for the number of data words in the token to be modified, either by removing certain data words or by adding new ones. In some cases tokens are removed entirely from the token stream.

In most applications, pipeline stages will typically only decode (be activated by) a few tokens; the stage does not recognize other tokens and passes them on unaltered. In a large number of cases, only one token is decoded: the data token word itself.

In many applications, the operation of a particular stage will depend upon the results of its own past operations. The "state" of the stage thus depends on its previous states. In other words, the stage depends upon stored state information, which is another way of saying it must retain some information about its own history one or more clock cycles ago. The invention is well-suited for use in pipelines that include such "state machine" stages as well as for the applications in which the latches in the data path are simple pipeline latches.

The suitability of the two-wire interface according to this example in such "state machine" circuits is a significant advantage of the example. Especially where a data path is being controlled by a state machine, the twowire interface technique above-described may be used to ensure that the "current state" of the machine stays in step with the data which it is controlling in the pipeline.

Fig. 6 is a simplified block diagram of an example of circuitry included in a pipeline stage for decoding a token address field; this illustrates a pipeline stage that has the characteristics of a "state machine". Each word of a token includes an "extension bit" which is HIGH if there are more words in the token or LOW if this is the last word of the token. If this is the last word of a token then the next valid data word is the start of a new token and therefore its address must be decoded. The decision as to whether or not to decode the token address in any given word thus depends upon knowing the value of the previous extension bit.

For the sake of simplicity only, the two-wire interface (with the acceptance and validation signals and latches) is not illustrated and all details to do with resetting the circuit are omitted. As before, an 8-bit data word is assumed by way of example only.

This exemplifying pipeline stage delays the data bits and the extension bit by one pipeline stage. It also decodes the DATA token and at the point when the first word of the DATA token is presented at the output of the circuit the signal "DATA_ADDR" is created and set HIGH. The data bits are delayed by the latches LDIN and LDOUT, each of which is repeated eight times for the eight data bits used in this example (corresponding to an 8-input, 8-output latch). Similarly the extension bit is delayed by extension bit latches LEIN and LEOUT.

In this example, the latch LEPREV is provided to store the most recent state of the extension bit. The value of the extension bit is loaded into LEIN and is then loaded into LEOUT on the next rising edge of the nonoverlapping clock phase signal PH1. Latch LEOUT thus contains the value of the current extension bit, but only during the second half of the non-overlapping, two-phase clock. Latch LEPREV, however, loads this extension bit value on the next rising edge of the clock signal PH0, that is, the same signal that enables the extension bit input latch LEIN. The output QEPREV of the latch LEPREV thus will hold the value of the extension bit during the previous PH0 clock phase.

The five bits of the data word output from the inverting Q output, plus the non-inverted MD[2], of the latch LDIN are combined with the previous extension bit value QEPREV in a series of logic gates NAND1, NAND2, and NOR1, whose operations are well known in the field of digital design. The designation "N_MD[m] indicates the logical inverse of bit m of the mid-data word MD[7:0]. Using known techniques of Boolean algebra it can be shown that the output signal SA from this logic block (the output from NOR1) is HIGH (a "1") only when the previous extension bit is a "0" (QPREV = "0") and the data word at the output of the non-inverting Q latch (the original input word) LDIN has the structure "000001xx", that is, the five high-order bits MD[7]-MD[3] bits are all "0" and the bit MD[2] is a "1" and the bits in the Zero-one positions have any arbitrary value.

There are thus four possible data words (there are four permutations of "xx") that will cause SA, and, thus the output of the address signal latch LADDR to whose input SA is connected, to become HIGH. In other words, this stage provides an activation signal (DATA_ADDR = "1") only when one of the four possible proper tokens is presented and only when the previous extension bit was a zero, that is, the previous data word was the last word in the previous series of token words, which means in turn that the present token word is the first one in the present token.

When the signal QPREV from latch LEPREV is LOW, the value at the output of the latch LDIN is therefore the first word of a new token. The gates NAND1, NAND2, and NOR1 decode the DATA token (000001xx). This address decoding signal SA is, however, then delayed in latch LADDR so that the signal DATA_ADDR has the same timing as the output data OUT_DATA and OUT_EXTN.

Fig. 7 is another simple example of a state-dependent pipeline stage that generates the signal LAST_OUT_EXTN to indicate the value of the previous output extension bit OUT_EXTN. One of the two enabling signals (at the CK inputs) to the present and last extension bit latches, LEOUT and LEPREV, respectively, is derived from the gate AND1 such that these latches only load a new value for the when the data is valid and is being accepted (the Q outputs are HIGH from the output validation and acceptance latches LVOUT and LAOUT, respectively). In this way, they only hold valid extension bits and are not loaded with spurious values associated with data that is not valid. In the embodiment shown in Fig. 7, the two-wire valid/accept logic includes the OR1 and OR2 gates with input signals consisting of the downstream acceptance signals and the inverting output of the validation latches LVIN and LVOUT, respectively. This illustrates one way in which the gates NAND1/2 and INV1/2 in Fig. 4 can be replaced if the latches have inverting outputs.

Although this is an extremely simple example of a "state-dependent" pipeline stage, since it depends on the state of only a single bit, it is generally true as in this example that all latches holding state information will be updated only when data is actually transferred between pipeline stages, that is, only when the data is both valid and being accepted by the next stage. Care must be taken to ensure that such latches are properly reset.

The generation and use of tokens according to this embodiment thus provide several advantages over known encoding techniques for data transfer through a pipeline.

First, the tokens as described above allow for variable length address field (and can utilize Huffman coding) to provide efficient representation of common tokens.

Second, consistent encoding of the length of a token allows the end of a token (and hence the start of the next) to be processed correctly (including simple non-manipulative transfer), even if the token is not recognized by the token decoder circuitry in a given pipeline stage.

Third, rules and hardware structures for the handling of unrecognized tokens (that is, for passing them on unmodified) allow communication between one stage and a downstream stage that is not its nearest neighbor in the pipeline. This also increases the expandability and efficient adaptability of the pipeline since it allows for future changes in the token set without requiring largescale redesigning of existing pipeline stages. The tokens are particularly useful when used in conjunction with the two-wire interface that is described above and below.

Figs. 8a and 8b, taken together (and referred to collectively below as Fig. 8), form a block diagram of a pipeline stage whose function is as follows: if the stage is processing a predetermined token (known in this example as the DATA token), then it will duplicate every word in this token with the exception of the first one, which includes the address field of the DATA token. If, on the other hand, the stage is processing any other kind of token then it will delete every word. The overall effect is that, at the output, only DATA tokens appear and each word within these tokens is repeated twice.

Many of the components of this illustrated system may be the same as those described in the much simpler structures shown in Figs. 4, 6, and 7. This illustrates a significant advantage : more complicated pipeline stages will still enjoy the same benefits of flexibility and elasticity, since the same two-wire interface may be used with little or no adaptation.

The data duplication stage shown in Fig. 8 is merely one example of the endless number of different types of operations that a pipeline stage could perform in any given application. This "duplication stage" illustrates, however, a stage that can form a "bottleneck", so that the pipeline according to this embodiment will "pack together".

A "bottleneck" can be any stage that either takes a relatively long time to perform its operations, or that creates more data in the pipeline than it receives. This example also illustrates that the two-wire accept/valid interface according to this embodiment can be adapted very easily to different applications.

The duplication stage shown in Fig. 8 also has two latches LEIN and LEOUT that, as in the example shown in Fig. 6, latch the state of the extension bit at the input and at the output of the stage, respectively. As Fig. 8a shows, the input extension latch LEIN is clocked synchronously with the input data latch LDIN and the validation signal IN_VALID.

For easo of reference, the various latches included in the duplication stage are paired below with their respective output signals:

Latch Output Labels

Latch	Output
LDIN	MID_DATA
LDOUT	OUT_DATA
LEIN	QIN
LEOUT	OUTEXTN
LAIN	QAIN
LAOUT	QAOUT
LI1	QI1
LO1	QO1
LI2	QI2
LO2	QO2 = DATA_TOKEN
LI3	QI3
LO3	QO3 = NOT_DUPLICATE

In the duplication stage, the output from the data latch LDIN forms intermediate data referred to as MID_DATA. This intermediate data word is loaded into the data output latch LDOUT only when an intermediate acceptance signal (labeled "MID_ACCEPT" in Fig. 8a) is set HIGH.

The portion of the circuitry shown in Fig. 8 below the acceptance latches LAIN, LAOUT shows the circuits that are added to the basic pipeline structure to generate the various internal control signals used to duplicate data. These include a "DATA_TOKEN" signal that indicates that the circuitry is currently processing a valid data token and a NOT_DUPLICATE signal which is used to control duplication of data. When the circuitry is processing a data token, the NOT_DUPLICATE signal toggles between a HIGH and a LOW state and this causes each word in the token to be duplicated once (but no more times). When the circuitry is not processing a valid data token then the NOT_DUPLICATE signal is held in a HIGH state, which means that the token words that are being processed are not duplicated.

As Fig. 8a shows, the upper six bits of 8-bit intermediate data word and the output signal QI1 from the latch LI1 form inputs to a group of logic gates NOR1, NOR2, NAND18. The output signal from the gate NAND18 is labeled S1. Using well-known Boolean algebra, it can be shown that the signal S1 is a "0" only when the output signal QI1 is a "1" and the MID_DATA word has the following structure: "000001xx", that is, the upper five bits are all "0", the bit MID_DATA[2] is a "1" and the bits in the MID_DATA[1] and MID_DATA[0] positions have any arbitrary value. Signal S1 therefore acts as a "token identification signal" which is low only when the MID_DATA signal has a predetermined structure and the output from the latch LI1 is a "1". The nature of the latch LI1 and its output QI1 is explained below.

Latch LO1 performs the function of latching the last value of the intermediate extension bit (labeled "MID_EXTN" and as signal S4), and it loads this value on the next rising edge of the clock phase PH0 into the latch LI1, whose output is the bit QI1 that is one of the inputs to the token decoding logic group that forms signal S1. Signal S1, as is explained above, may only drop to a "0" if the signal QI1 is a "1" (and the MID_DATA signal has the predetermined structure). Signal S1 may therefore only drop to a "0" whenever the last extension bit was "0", indicating that the previous token has ended, and therefore the MID_DATA word is the first data word in a new token.

The latches LO2 and LI2 together with the NAND gates NAND20 and NAND22 together form storage for the signal DATA_TOKEN. In the normal situation, the signal QI1 at the input to NAND20 and the signal S1 at the input to NAND22 will both be at logic "1". It can be shown, again by the techniques of Boolean algebra, that in this situation these NAND gates operate in the manner of inverters, that is, the signal QI2 from the output of latch LI2 is inverted in NAND20 and then this signal is inverted again by NAND22 to form the signal S2. In this case, since there are two logical inversions in this path, the signal S2 will have the same value as QI2.

It can also be seen that the signal DATA_TOKEN at the output of latch LO2 forms the input to latch LI2. As a result, as long as the situation in which both QI1 and S1 are HIGH remains the signal DATA_TOKEN will retain its state (whether "0" or "1"). This is true even though the clock signals PH0 and PH1 are clocking the latches (LI2 and LO2 respectively). The value of DATA_TOKEN can only change when one or both of the signals QI1 and S1 are "0".

As explained earlier, the signal QI1 will be "0" when the previous extension bit was "0". Thus it will be "0" whenever the MID_DATA value is the first word of a token (and thus includes the address field for the token). In this situation, the signal S1 may be either "0" or "1". As explained earlier, signal S1 will be "0" if the MID_DATA word has the predetermined structure that in this example indicates a "DATA" token. If the MID_DATA word has any other structure, (indicating that the token is some other token, not a DATA token), S1 will be "1".

If QI1 is "0" and S1 is "1", indicating some token other than a data token, then as is well known in the field of digital electronics, the output of NAND20 will be "1". The NAND gate NAND22 will invert this (as previously explained) and the signal S2 will thus be a "0". As a result, this "0" value will be loaded into latch LO2 at the start of the next PH1 clock phase and the DATA_TOKEN signal will become "0", indicating that the circuitry is not processing a DATA token.

If QI1 is "0" and S0 is "0", thereby indicating a DATA token, then the signal S2 will be "1" (regardless of the other input to NAND22 from the output of NAND20). As a result, this "1" value will be loaded into latch LO2 at the start of the next PH1 clock phase and the DATA_TOKEN signal will become "1", indicating that the circuitry is processing a DATA token.

The NOT_DUPLICATE signal (the output signal QO3) is similarly loaded into the latch LI3 on the next rising edge of the clock PH0. The output signal QI3 from the latch LI3 is combined with the output signal QI2 in a gate NAND24 to form the signal S3. As before, Boolean algebra can be used to show that the signal S3 is a "0" only when both of the signals QI2 and QI3 have the value "1". If the signal QI2 becomes a "0", that is, the DATA TOKEN signal is a "0", then the signal S3 becomes a "1". In other words, if there is not a valid DATA TOKEN (QI2 = 0) or the data word is not a duplicate (QI3 = 0), then the signal S3 goes high.

Assume now that the data token signal remains HIGH for more than one clock signal. Since the NOT_DUPLICATE signal (QO3) is "fed back" to the latch LI3 and will be inverted by the gate NAND 24 (since its other input QI2 is held HIGH), the output signal QO3 will toggle between "0" and "1". If there is no valid data token, however, the signal QI2 will be a "0", and the signal S3 and, therefore, the output QO3, will be forced HIGH until the DATA_TOKEN signal once again goes to a "1".

The output QO3 (the NOT_DUPLICATE signal) is also fed back and is combined with the output QA1 from the acceptance latch LAIN in a series of logic gates (NAND16 and INV16, which together form an AND gate) that have as their output a "1" only when the signals QA1 and QO3 both have the value "1". As Fig. 8a shows, the output from the AND gate (the gate NAND16 followed by the gate INV16) also forms the acceptance signal IN_ACCEPT, which is used as above in the two-wire interface structure.

The acceptance signal IN_ACCEPT is also used as an enabling signal to the latches LDIN, LEIN, and LVIN. The result of this is that if the NOT_DUPLICATE signal is low, then the acceptance signal IN_ACCEPT will also be low, and all three of these latches will be disabled and will hold the values stored at their outputs. The stage will not accept new data until the NOT_DUPLICATE signal becomes HIGH, in addition to the requirements described above for forcing the output from the acceptance latch LAIN high.

As long as there is a valid DATA_TOKEN (the DATA_TOKEN signal QO2 is a "1"), the signal QO3 will toggle between the HIGH and LOW states, so that the input latches will be enabled and will be able to accept data at most during every other complete cycle of both clock phases PH0, PH1. The additional condition that the following stage be prepared to accept data, as indicated by a "HIGH" OUT_ACCEPT signal, must, of course, still be satisfied. The output latch LDOUT will, therefore, place the same data word onto the output bus OUT_DATA for at least two full clock cycles. The OUT_VALID signal will be a "1" only when there is both a valid DATA_TOKEN (QO2 HIGH) and the validation signal QVOUT is HIGH.

The signal QEIN, which is the extension bit corresponding to MID_DATA, is combined with the signal S3 in a series of logic gates (INV10 and NAND10) to form a signal S4. During a DATA token, each data word MID_DATA will be repeated by loading it into the output latch LDOUT twice. During the first of these, S4 will be forced to a "1" by the action of NAND10. The signal S4 is loaded in the latch LEOUT to form QUTEXTN at the same time as MID_DATA is loaded into LDOUT to form OUT_DATA[7:0].

Thus, the first time a given MID_DATA is loaded into LEOUT, the associated OUTEXTN will be forced high, whereas, on the second occasion, OUTEXTN will be the same as the signal QEIN. Now consider the situation during the very last word of a token in which QEIN is known to be low. During the first time MID_DATA is loaded into LDOUT, OUTEXTN will be "1" and the second time OUTEXTN will be "0", indicating the true end of the token.

The output signal QVIN from the validation latch LVIN is combined with the signal QI3 in a similar gate combination (INV12 and NAND12) to form a signal S5. Using known Boolean techniques, it can be shown that the signal S5 is HIGH either when the validation signal QVIN is HIGH, or when the signal QI3 is low (indicating that the data is a duplicate). The signal S5 is loaded into the validation output latch LVOUT at the same time that MID_DATA is loaded into LDOUT and the intermediate extension bit (signal S4) is loaded into LEOUT. Signal S5 is also combined with the signal QO2 (the data token signal) in the logic gates NAND30. and INV30 to form the output validation signal OUT_VALID. As was mentioned earlier, OUT_VALID is HIGH only when there is a valid token and the validation signal QVOUT is high.

The MID_ACCEPT signal is combined with the signal S5 in a series of logic gates (NAND26 and INV26) that performs the well-known AND function to form a signal S6 that is used as one of the two enabling signals to the latches LO1, LO2 and LO3. The signal S6 rises to a "1" when the MID_ACCEPT signal is HIGH and when either the validation signal QVIN is high, or when the token is a duplicate (QI3 is a "0"). If the signal MID_ACCEPT is HIGH, the latches LO1-LO3 will therefore be enabled when the clock signal PH1 is high whenever valid input data is loaded at the input of the stage, or when the latched data is a duplicate.

From the discussion above, one will see that the stage shown in Figs. 8a and 8b will receive and transfer data between stages under the control of the validation and acceptance signals, as in previous embodiments, with the exception that the output signal from the acceptance latch LAIN at the input side is combined with the toggling duplication signal so that a data word will be output twice before a new word will be accepted.

The various logic gates such as NAND16 and INV16 may, of course, be replaced by equivalent logic circuitry (in this case, a single AND gate). Similarly, if the latches LEIN and LVIN, for example, have inverting outputs, the inverters INV10 and INV12 would not be necessary; rather, the corresponding input to the gates NAND10 and NAND12 could be tied directly to the inverting outputs of these latches. As long as the proper logical operation is performed, the stage will operate in the same manner. Data words and extension bits will still be duplicated.

One should note that the duplication function that the illustrated stage performs will not be performed unless the first data word of the token has a "1" in the third position of the word and "0's" in the five high-order bits. (Of course, the required pattern can be easily changed and set by selecting other logic gates and interconnections than the NOR1, NOR2, NND18 gates shown.)

In addition, as Fig. 8 shows, the OUT_VALID signal will be forced low during the entire token unless the first data word has the structure described above. This has the effect that all tokens except the one that causes the duplication process will be deleted from the token stream, since a device connected to the output terminals (OUTDATA, OUTEXTN and OUTVALID) will not recognize these token words as valid data.

As before, both validation latches LVIN, LVOUT in the stage can be reset by a single conductor NOT_RESET0, and a single resetting input R on the downstream latch LVOUT, with the reset signal being propagated backwards to cause the upstream validation latch to be forced low on the next clock cycle.

It should be noted that in the example shown in Fig. 8 the duplication of data contained in DATA tokens serves only as an example of the way in which circuitry may manipulate the ACCEPT and VALID signals so that more data leaves the pipeline stage than arrives at the input. Similarly, the example in Fig. 8 removes all non-DATA tokens purely as an illustration of the way in which circuitry may manipulate the VALID signal to remove data from the stream. In most typical applications, however, a pipeline stage will simply pass on any tokens that it does not recognize unmodified so that other stages further along in the pipeline may act upon them if required.

Figs. 9a and 9b together show an example of a timing diagram for the data duplication circuit shown in Figs. 8a and 8b. As before, the timing diagram shows the relationship between the two-phase clock signals, the various internal and external control signals, and the manner in which data is clocked between the input and output sides of the stage and is duplicated.

Claims

1. A data pipeline system for processing data including a plurality of sequential pipeline stages, each having an input to receive an input signal (IN_DATA), an output to deliver an output signal (OUT_DATA) to the input of a following stage and at least one data storage device (LDOUT);
   CHARACTERIZED in that:
   validation circuitry (LVIN, LVOUT) is included in each stage for generating a validation signal (OUT_VALID) with a first state when data stored in the stage is valid and with a second state when data stored in the stage is invalid;
   each adjacent pair of pipeline stages is connected by an acceptance line for conveying an acceptance signal (IN_ACCEPT, OUT_ACCEPT) indicative of the ability of the following pipeline stage to load data stored in the current pipeline stage; and
   enabling circuitry is provided responsive to the validation signal (OUT_VALID) and the acceptance

signal (IN__ACCEPT, OUT__ACCEPT) for generating an enabling signal to enable loading of data into the storage device.

2. A system according to claim 1, including:
an output data storage device (LDOUT); and
at least one validation storage device (LVOUT) that stores the validation signal for its stage and is included in the validation circuitry.

3. A system according to claim 2, wherein the validation storage device (LVOUT) is connected to the enabling circuitry.

4. A system according to any one of the preceding claims, and arranged such that the acceptance signal (IN__ACCEPT, OUT__ACCEPT) constitutes the enabling signal for the data storage device and validation circuitry.

5. A system according to claim 2 or 3, or to claim 4 when appended to claim 2, wherein:
the data storage devices include a primary data storage device and a secondary data storage device;
data is loaded into the respective primary data storage devices and validation signals are loaded into respective primary validation storage devices at the same time; and
data is loaded into each respective primary data storage device when the acceptance signal assumes an enabling state.

6. A system according to claim 5, wherein the acceptance signal is arranged to assume the enabling state only when the acceptance signal associated with the immediately following data storage device is in the enabling state or the data in the data storage device is invalid.

7. A system according to claim 5 or 6, and including acceptance processing circuitry.

8. A system according to any one of the preceding claims, wherein:
an input data storage device (LDIN) of each stage forms a secondary data storage device and a secondary validation storage device (LVIN) is included in each stage; and
a primary acceptance storage device (LAOUT) is included in each stage for storing the state of the acceptance signal (OUT__ACCEPT) of an immediately following stage.

9. A system according to claim 8, wherein the primary data output signals and validation output signals are arranged to be derived from a single primary storage unit and the secondary data output signals and validation output signals are arranged to be formed as sections of a single secondary storage unit.

10. A system according to claim 8 or 9, wherein: a secondary acceptance storage device (LAIN) is included in each stage; and
each stage is connected to a multi-phase, nonoverlapping, clock (PH0, PH1).

11. A system according to claim 10, wherein: the primary data storage device (LDOUT), primary validation storage device (LVOUT) and secondary acceptance storage device (LAIN) is arranged to be enabled by a first clock phase signal (PH1) and the secondary data storage device (LDOUT), secondary validation storage device (LVOUT) and primary acceptance storage device (LAIN) is arranged to be enabled by a second clock phase signal (PH0).

12. A system according to any one of the preceding claims, wherein:
each pipeline stage includes predetermined processing circuitry; and
the output from each secondary data storage device (LDIN) is connected as an input to the primary data storage device (LDOUT) of the corresponding stage via arbitrary logic circuitry.

13. A system according to claim 12, wherein the predetermined processing circuitry has at least one active mode and an inactive mode.

14. A system according to claim 12 or 13, wherein:
predetermined ones of the pipeline stages include a decoding circuit connected to the output of a predetermined one of the data storage devices, whereby the processing circuitry of each pipeline stage assumes the active state only when the stage contains a predetermined stage activation signal pattern and remains in the activation mode until the stage contains a predetermined stage deactivation signal pattern.

15. A system according to claim 10 or 14, wherein:
each pipeline stage includes a present extension bit input latch (LEIN) and an extension bit output latch (LEOUT) for loading an extension bit under control of the first clock phase signal (PH0), with the extension bit being transferred from a preceding device via an extension bit conductor (IN__EXTN; OUT__EXTN);
the output from the present extension bit input latch (LEIN) is connected to the input of the extension bit output latch (LEOUT);
loading of the extension bit into the present extension bit input latch (LEIN) is enabled by the first clock signal (PH0) and loading of the extension bit into the extension bit output latch is enabled by the first clock signal (PH1), whereby the extension bit output latch loads the value of the extension bit previously loaded into the extension bit input latch;
each pipeline stage includes a previous extension bit latch (LEPREV) for loading the output from the extension bit output latch (LEOUT) under control of the first clock signal (PH0);
the extension bit is transferred from the preceding device via an extension bit conductor (IN__EXTN; OUT__EXTN) to the input of the present extension bit latch (LEIN); and
the processing circuitry of each pipeline stage assumes the active state only when previous extension bit, loaded in the previous extension bit latch (LEPREV) is in a predetermined one of two logical states.

16. A data pipeline system for processing data including a plurality of sequential pipeline stages, there being an input data storage device (LDIN) and an output data storage device (LDOUT) in each stage, with the output data storage device of each stage connected to the following input data storage device;
CHARACTERIZED in that:
each pipeline stage is arranged to generate an acceptance signal to the immediately preceding pipeline stage with a first state when it does not contain valid data and when it contains valid data that can be transferred to a following data storage device, and with a second state when it contains valid data that cannot be passed to the following data storage device.

17. A system according to claim 16, wherein:
each stage has an unblocked state, in which it is able to receive data without loss of previously stored valid data, and a blocked state, in which it contains valid data that cannot be transferred from the corresponding data storage device;
data can be transferred into a current one of the pipeline stages even when at least one other pipeline stage following the current stage is in the blocked state; and
each stage includes predetermined processing circuitry with an active state, which it enters when data entering the stage has a predetermined activation pattern, and an inactive state, in which the stage passes data to the following stage without processing.

18. A data pipeline system for processing data including a plurality of sequential pipeline stages, there being an input data storage device (LDIN) and an output data storage device (LDOUT) in each stage, with the output data storage device of each stage connected to the following input data storage device;
CHARACTERIZED in that:
each stage has an unblocked state, in which it is able to receive data without loss of previously stored valid data, and a blocked state, in which it contains valid data that cannot be transferred from the corresponding data storage device; and
data can be transferred into a current one of the pipeline stages even when at least one other pipeline stage following the current stage is in the blocked state.

19. A system according to claim 18, wherein :
each pipeline stage is operable to generate an acceptance signal to the immediately preceding pipeline stage with a first state when it does not contain valid data and when it contains valid data that

can be transferred to a following data storage device, and with a second state when it contains valid data that cannot be passed to the following data storage device; and

each stage includes predetermined processing circuitry with an active state, which it enters when data entering the stage has a predetermined activation pattern, and an inactive state, in which the stage passes data to the following stage without processing.

20. A data pipeline system for processing data including a plurality of sequential pipeline stages, there being an input data storage device (LDIN) and an output data storage device (LDOUT) in each stage, with the output data storage device of each stage connected to the following input data storage device;

CHARACTERIZED in that:

each stage includes predetermined processing circuitry with an active state, which it enters when data entering the stage has a predetermined activation pattern, and an inactive state, in which the stage passes data to the following stage without processing.

21. A system according to claim 20, wherein:

each stage has an unblocked state, in which it is able to receive data without loss of previously stored valid data, and a blocked state, in which it contains valid data that cannot be transferred from the corresponding data storage device;

data can be transferred into a current one of the pipeline stages even when at least one other pipeline stage following the current stage is in the blocked state; and

each pipeline stage generates an acceptance signal to the immediately preceding pipeline stage with a first state when it does not contain valid data and when it contains valid data that can be transferred to a following data storage device, and with a second state when it contains valid data that cannot be passed to the following data storage device.

22. A method for encoding digital data in a pipeline, with the pipeline including a plurality of stages and each stage having an active mode, in which it transforms corresponding work data, and a passive mode;

CHARACTERIZED by the following steps:

a) applying a series of data words as a data stream to a first one of the pipeline stages in the form of digital signals;

b) generating and assigning to predetermined ones of the pipeline stages respective, predetermined, unique data activation words, whereby each stage assumes its active state only upon receipt of the corresponding data activation word;

c) for a plurality of input data blocks, including in the data stream a sequence of address signals and, for each data word, an extension bit, a series of address bits, and data bits, whereby the extension bit has a first and a second logical state;

d) setting the extension bit to the first logical state for a selected boundary word in each data block and to the second logical state for every other data word in the data block; and

e) setting the address bits equal to corresponding bits of the activation code for the pipeline stage for which the data bits in the same data block are work data for the corresponding stage.

23. In combination for providing a controlled passage of data,

a plurality of stages coupled to one another in a pipeline arrangement,

first means in each of the stages in the pipeline arrangement for indicating whether such stage is able to pass data to the next stage in the pipeline arrangement;

second means in each of the stages in the pipeline arrangement for indicating whether such stage is able to receive data from the previous stage in the pipeline arrangement, and

means responsive in each stage to the indications of the first means from such stage and the second means from the next stage in the pipeline arrangement for passing the data from such stage to the next stage in the pipeline arrangement in accordance with such indications.

24. In a combination as set forth in claim 23,

fourth means for providing clock signals, and

fifth means for synchronizing the operation of the third means in each stage with the clock signals in passing the data from such stage to the next stage in the pipeline arrangement.

25. In a combination as set forth in claim 24, wherein
the first means in each of the stage in the pipeline arrangement include first latching means, and
the second means in each of the stages in the pipeline arrangement include second latching means, and
the third means in each of the stages in the pipeline arrangement include third latching means.

26. In a combination as set forth in claim 25,
fourth means for providing clock signals having first and second phases,
the first latching means in each stage in the pipeline arrangement means being responsive to the first phase in the clock signals to become latched,
the third latching means in each stage in the pipeline arrangement being responsive to the first phase in the clock signals to become latched, and
the second latching means in each stage in the pipeline arrangement being responsive to the second phase in the clock signals to become latched.

27. In combination for providing a controlled passage of data,
a plurality of stages coupled to one another in a pipeline arrangement,
first means in each of the stages in the pipeline arrangement for storing data,
second means in each individual one of the stages in the pipeline arrangement for determining at each instant whether the next stage in the pipeline arrangement is able at that instant to receive a transfer to the next stage of the data stored in such individual stage at that instant, and
third means in each individual one of the stages in the pipeline arrangement for determining at each instant whether such stage is able at that instant to pass data to the next stage in the pipeline arrangements,
fourth means in each individual one of the stages in the pipeline arrangement for passing the data stored in such stage at each instant to the next stage in the pipeline arrangement in accordance with the determination provided at that instant by the third means for such individual stage and the second means for the next stage in the pipeline arrangement,
the operation of the fourth means for each stage in the pipeline arrangement at each instant being independent of the operation of the fourth means for the other stages in the pipeline arrangement at that instant to provide for the passage of data at each instant of data from individual ones of the stages in the pipeline arrangement at that instant to the stages next to such individual stages in the pipeline arrangement without the passage at such instant of data from other stages in the pipeline arrangement to the stages next to such other stages in the pipeline arrangement.

28. In a combination as set forth in claim 27, means for resetting at least alternate stages in the pipeline arrangement to provide for the subsequent passage of information to each stage from the previous stage in the pipeline arrangement.

29. In a combination as set forth in claim 27, fifth means for providing clock signals,
the first means in each stage in the pipeline arrangement including first latching means,
the second means in each stage in the pipeline arrangement including second latching means,
the third means in each stage in the pipeline arrangement including third latching means,
the first, second and third latching means being synchronized in operation in accordance with the clock signals from the fifth means.

30. In a combination as set forth in claim 27,
fifth means for providing an individual address for each of the different stages in the pipeline arrangement, each of the addresses being formed by an individual sequence of binary bits with a different number of binary bits in the sequence than the number of binary bits in the sequence for other stages in the pipeline arrangement, the combination of the binary bits for each progressive number of binary bits in each sequence being different from the combination of the binary bits for such progressive number of binary bits in the sequences for the other stages in the pipeline arrangement,
means in each stage in the pipeline arrangement for responding to the individual address for such stage, and
means for passing each of the addresses in the pipeline arrangement sequentially through the successive stages in the pipeline arrangement to the stage responsive in the pipeline arrangement to such individual address.

31. In combination for providing a controlled passage of data,
a plurality of stages coupled to one another in a pipeline arrangement,
first means for providing an individual address for each of the different stages in the pipeline arrangement, each of the individual addresses being formed by an individual sequence of binary bits with a different number of binary bits in the sequence than the number of binary bits in the sequence for other stages in the pipeline arrangement, the combination of the binary bits for each progressive number of binary bits in each sequence being different from the combination of the binary bits for such progressive number of binary bits in the sequence for the other stages in the pipeline arrangement,
second means in each stage in the pipeline arrangement for responding to the individual address for such stage, and
third means for passing each of the addresses in the pipeline arrangement sequentially through the successive stages in the pipeline arrangement to the stage responsive in the pipeline arrangement to such individual address.

32. In a combination as set forth in claim 31,
each of the individual addresses being associated in a token with data individual to such address, and
means responsive in each stage in the pipeline arrangement to the address individual to such stage for processing the data associated with such individual address.

33. In a combination as set forth in claim 22,
each of the addresses being included in a token,
each of the tokens being formed from an individual number of words, each of the words in each token having at a particular position an extension bit indicating the termination of the token or the extension of the token to the next word, and
means in each individual stage in the pipeline arrangement for responding to the extension bit in each successive word for a token having an address identified by such individual stage for terminating or extending such token in accordance with the characteristics of such extension bit in each such successive word in such token.

34. In a combination as set forth in claim 33, at least one of the words in each token including data, and
means responsive in each individual stage to the token having the address identified by such individual stage for processing the data in such token.

35. In a combination as set forth in claim 34,
means in each stage in the pipeline arrangement for providing a first indication of the readiness of such stage to pass data in such stage to a next stage in the pipeline arrangement and a second indication of the readiness of such stage to receive data from the previous stage in the pipeline arrangement, and
means in each stage in the pipeline arrangement for passing data to the next stage in the pipeline arrangement in accordance with the indication in such stage of the readiness of such stage to pass data to the next stage in the pipeline arrangement and the indication in the next stage in the pipeline arrangement of the readiness of such next stage to receive data.

36. In combination for processing data,
a plurality of stages connected in a pipeline arrangement,
first means for providing tokens each formed from a number of words between one (1) and a particular number greater than one (1), each of the words in each token including an extension bit and at least one of the words in each token including an address and words in the token including data, the address in each token identifying an individual one of the stages in the pipeline arrangement, the extension bit in each word in each token having first characteristics indicating the beginning of the token and second characteristics indicating the extension of the token to the word incorporating the extension bit,
second means for passing each token to the successive stages in the pipeline arrangement until the token reaches the individual stage identified in the pipeline arrangement by the address in that token,
third means in each stage in the pipeline arrangement for identifying the address individual to that stage,

fourth means in each stage in the pipeline arrangement for processing the data in each token upon the identification by that stage of the address individual to such stage.

37. In a combination as set forth in claim 36,
the fourth means in each stage in the pipeline arrangement including means for processing the data in the successive words in the tokens identified by such stage until the occurrence of an extension bit identifying, in one of the successive words, the beginning of the next token.

38. In a combination as set forth in claim 37,
fifth means for providing an individual address for each of the stages in the pipeline arrangement, each of the addresses being formed by an individual number of binary bits in the sequence than the number of binary bits in the sequence for other stages in the pipeline arrangement, the combination of binary bits for each progressive number of binary bits in each sequence being different from the combination of the binary bits for such progressive number of binary bits in the sequence for the other stages in the pipeline arrangement, and
sixth means in each individual stage in the pipeline arrangement for passing each of the token to the next stage in the pipeline arrangement when the address in such token is not identified by such individual stage.

39. In a combination as set forth in claim 37,
fifth means in each individual stage in the pipeline arrangement for determining at each instant whether the next stage in the pipeline arrangement is able at that instant to receive a transfer to the next stage of the data stored in such individual stage at that instant,
sixth means in each individual stage in the pipeline arrangement for determining at each instant whether such stage is able at that instant to pass data to the next stage in the pipeline arrangement, and
seventh means in each individual stage in the pipeline arrangement for passing data in such stage to the next stage in the pipeline arrangement in accordance with the determination provided at that instant by the sixth means for that stage and by the fifth means for the next stage in the pipeline arrangement.

FIG. 1

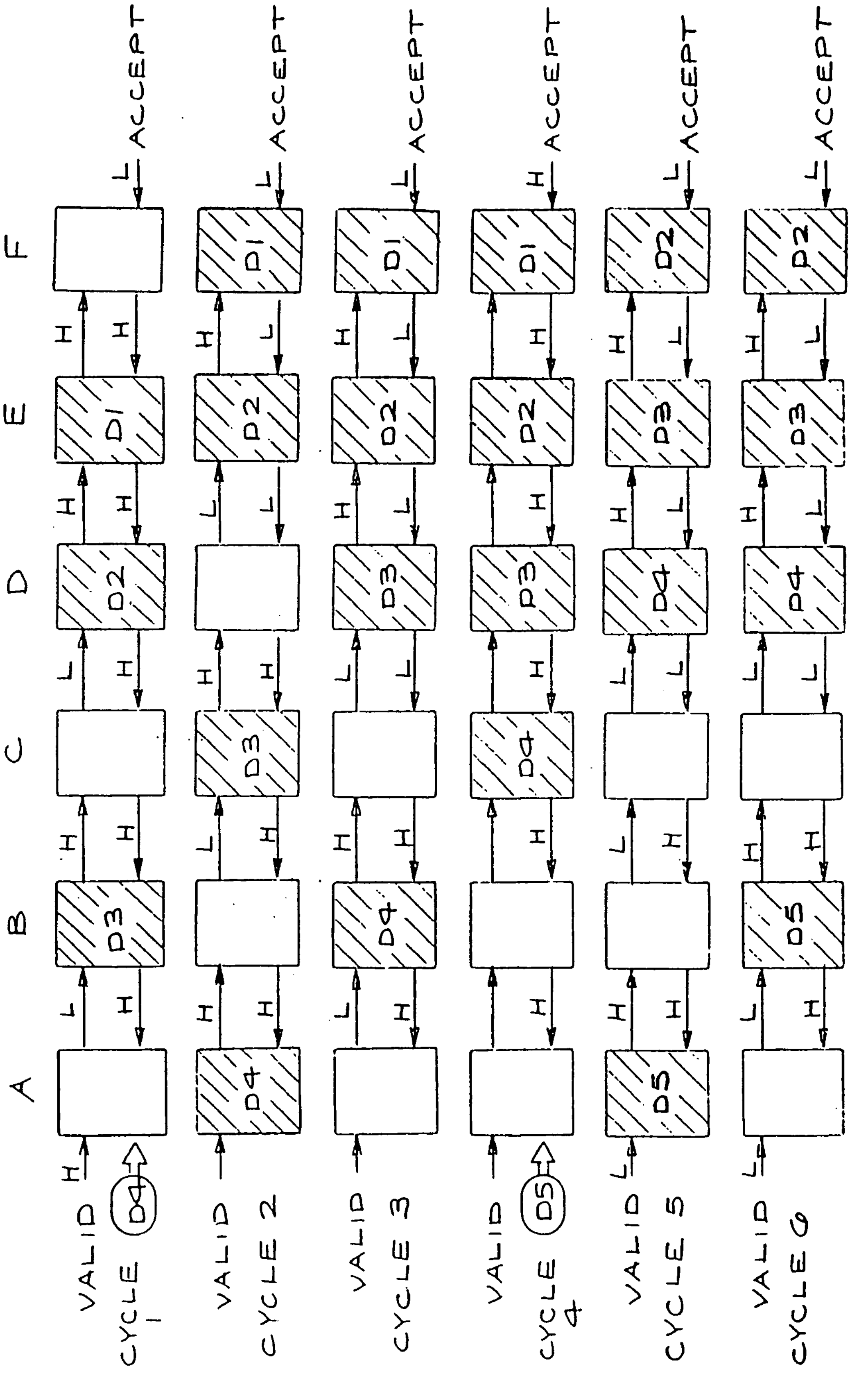
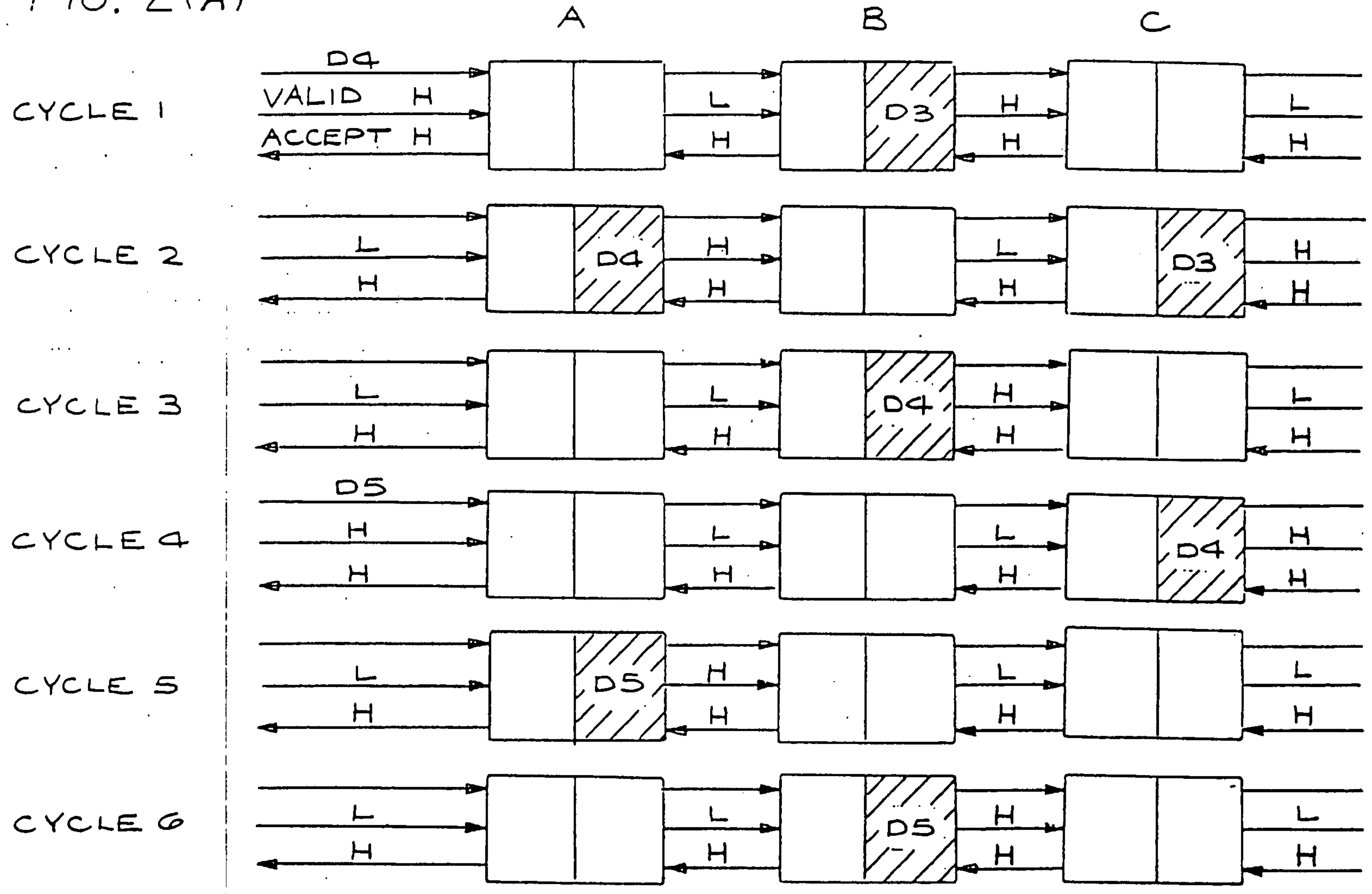

A
B
C
D
E
F
VALID
ACCEPT
CYCLE 1
CYCLE 2
CYCLE 3
CYCLE 4
CYCLE 5
CYCLE 6
D1
D2
D3
D4
D5
H
L

FIG. 2(A)


A
B
C
CYCLE 1
D4
VALID H
ACCEPT H
L
H
D3
H
H
L
H
CYCLE 2
L
H
D4
H
H
L
H
D3
H
H
CYCLE 3
L
H
L
H
D4
H
H
L
H
CYCLE 4
D5
H
H
L
H
L
H
D4
H
H
CYCLE 5
L
H
D5
H
H
L
H
L
H
CYCLE 6
L
H
L
H
D5
H
H
L
H

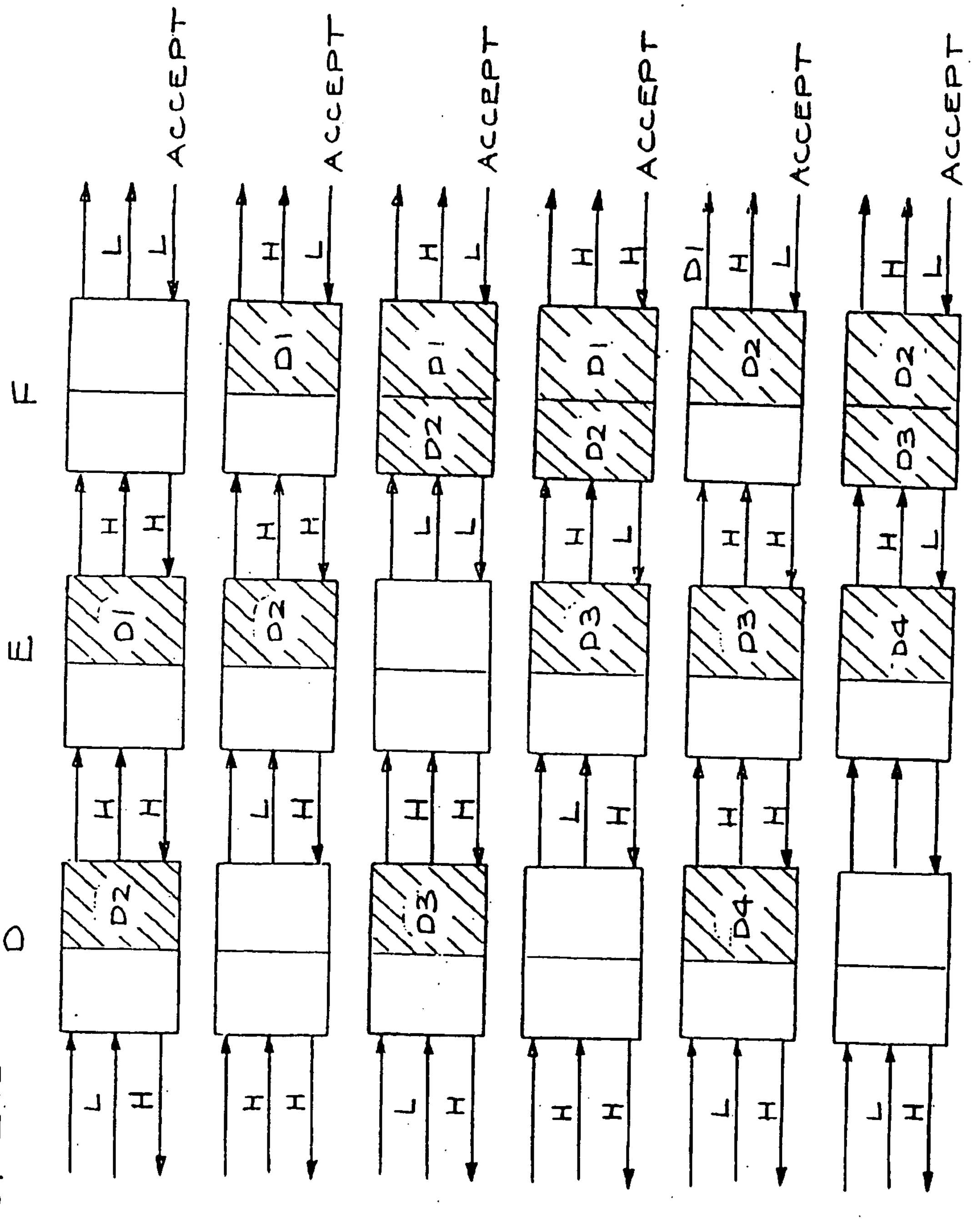


FIG. 2(B)

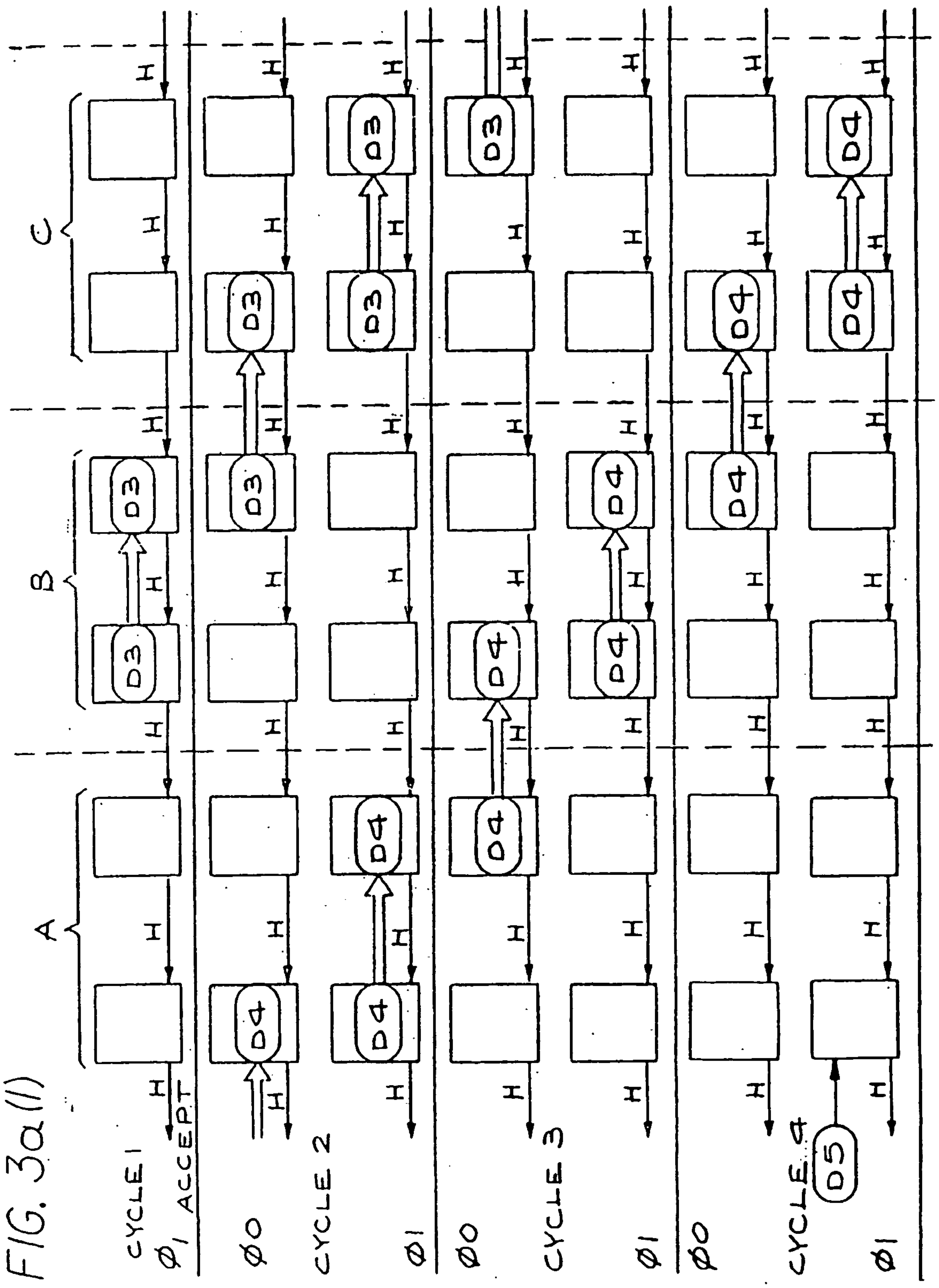
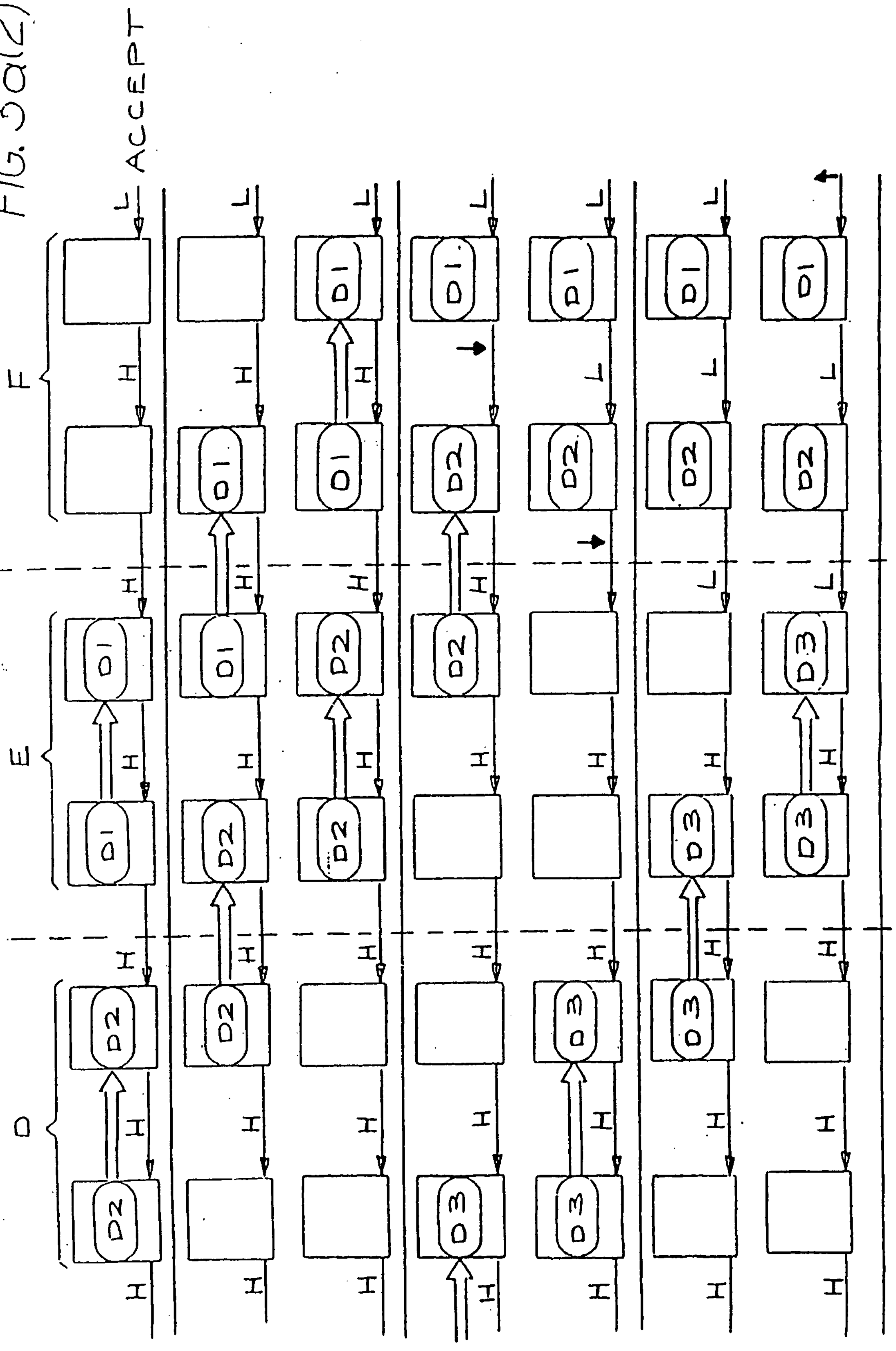

A
B
C
H
D3
D4
D5
CYCLE 1
ϕ_1 ACCEPT
ϕ_0
CYCLE 2
ϕ_1
ϕ_0
CYCLE 3
ϕ_1
ϕ_0
CYCLE 4
ϕ_1

FIG. 3a(1)

FIG. 3a(2)



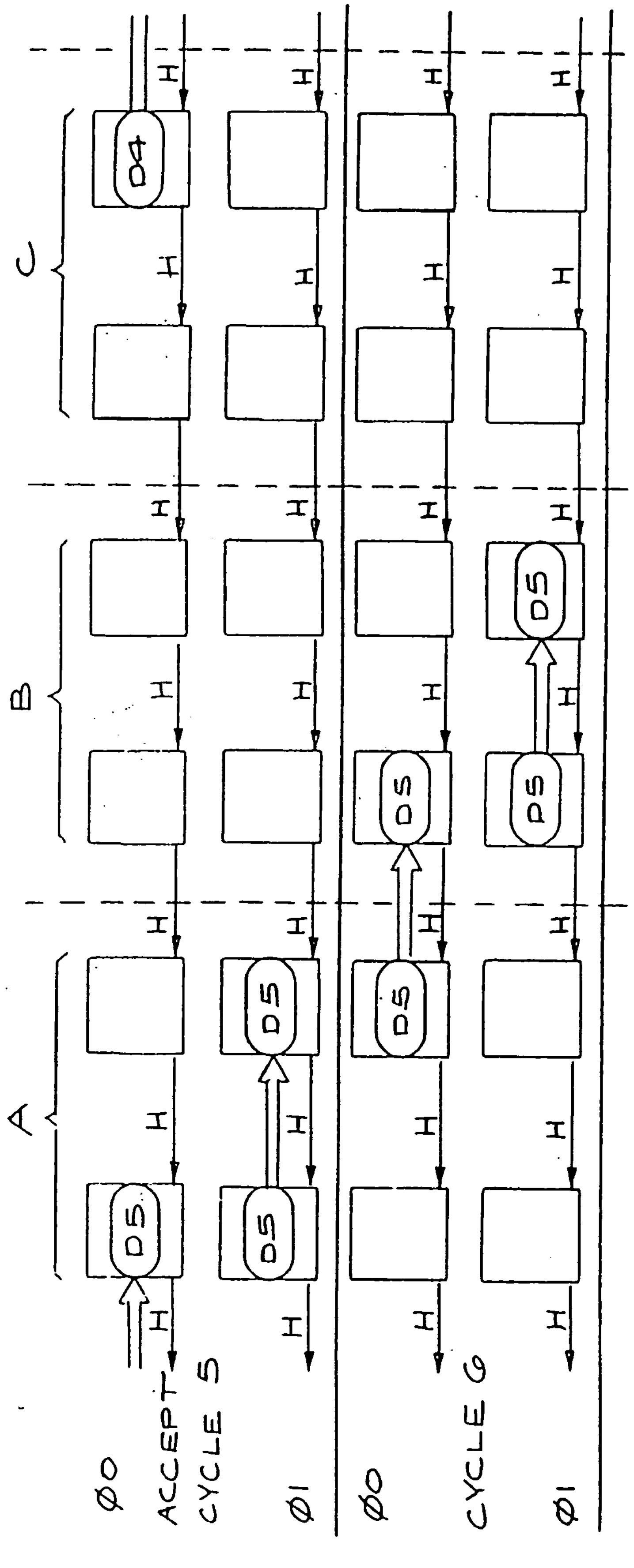


FIG. 3b(1)

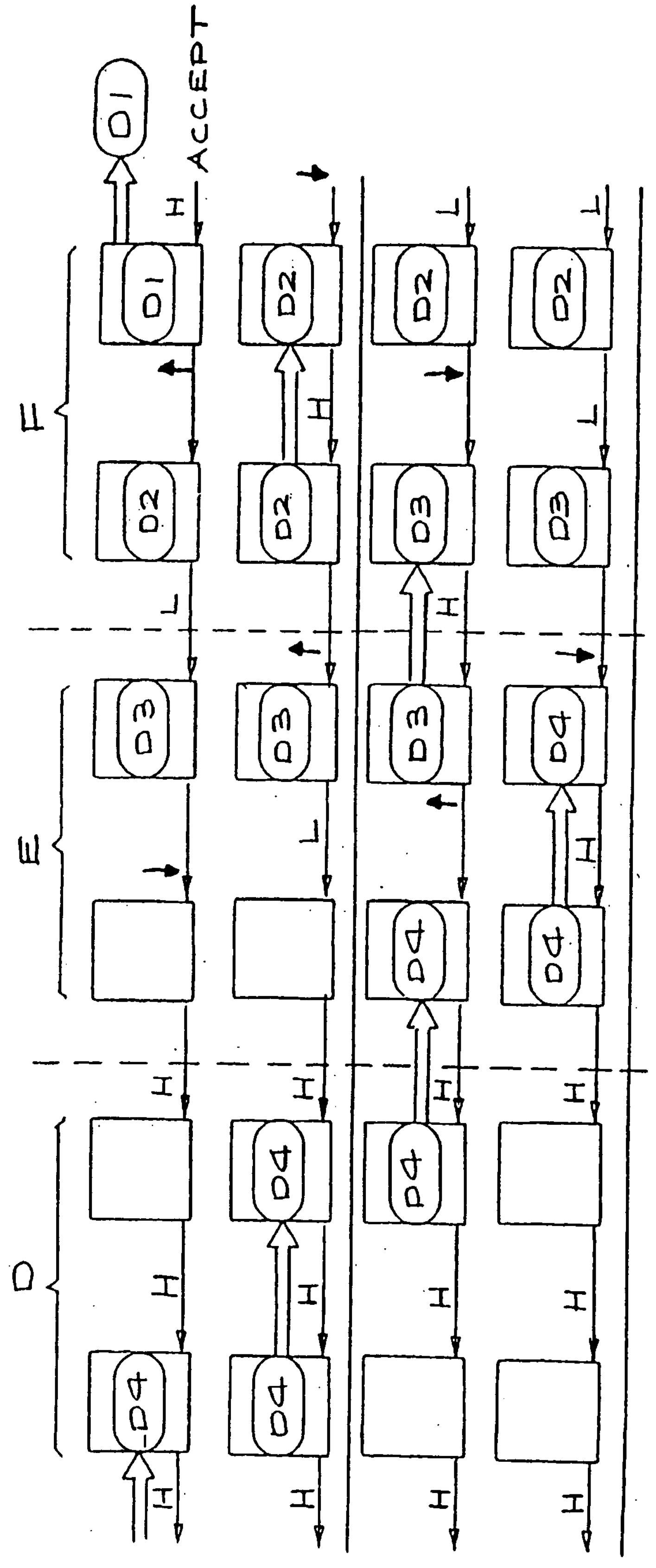
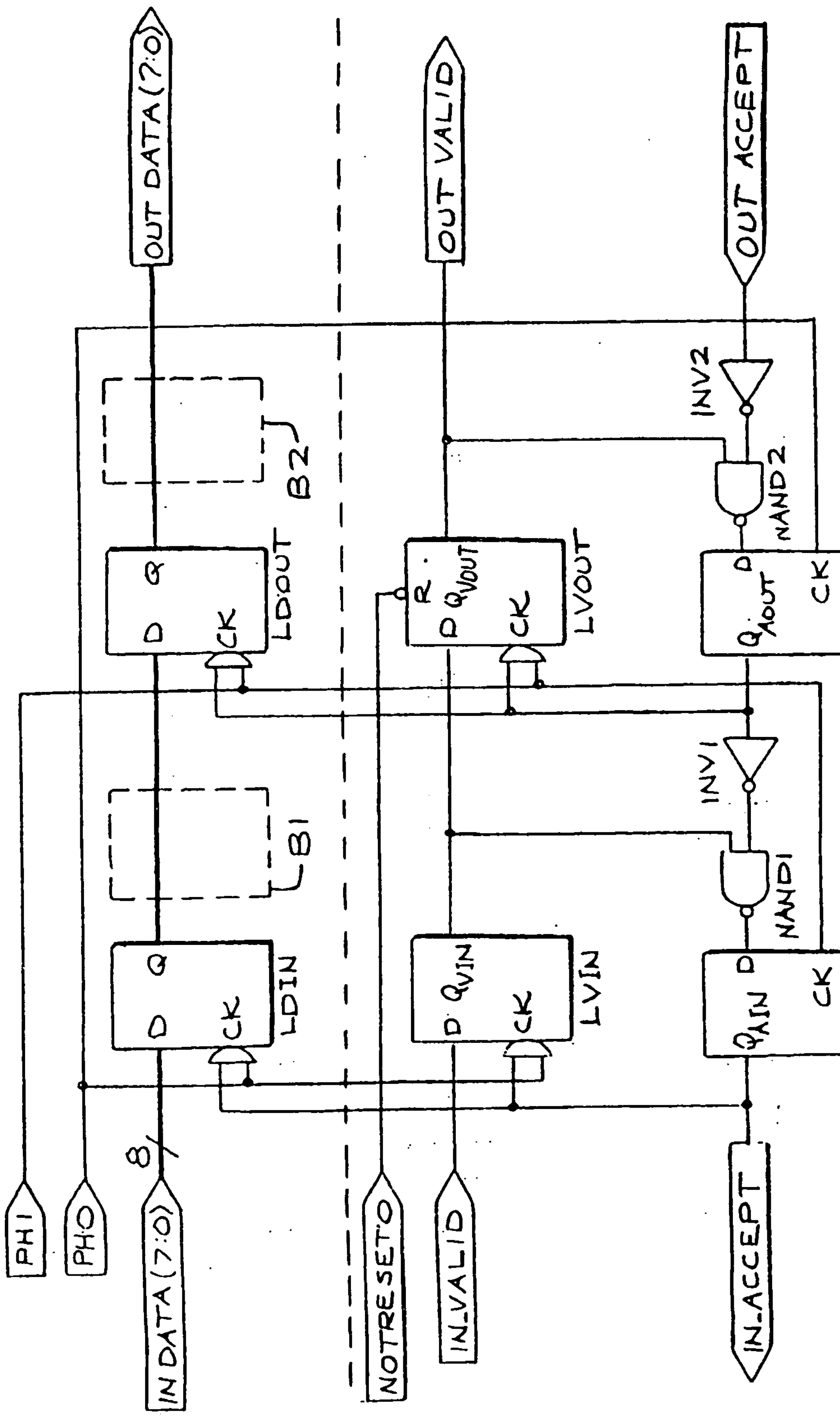


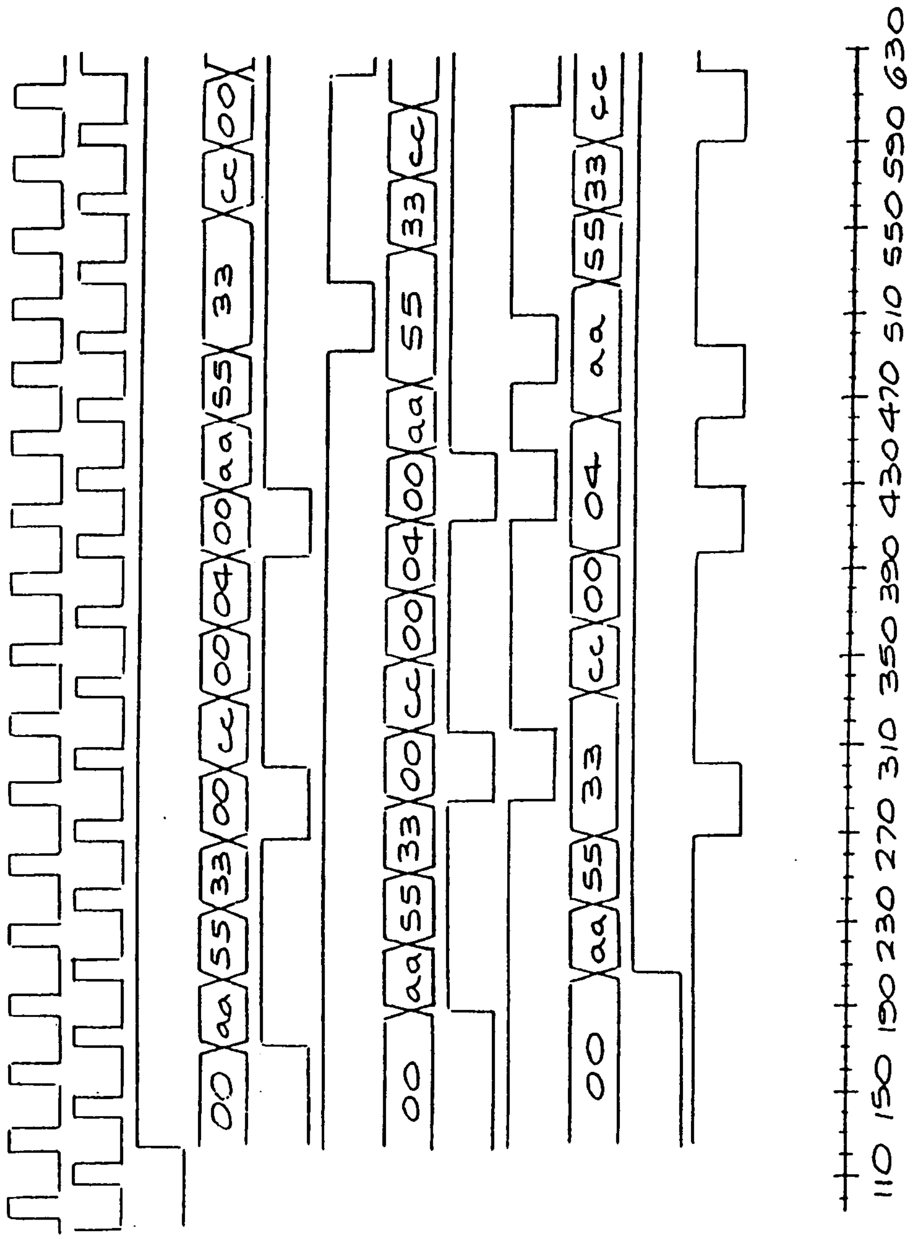
FIG. 3b(2)



OUT DATA(7:0)
OUT VALID
OUT ACCEPT
INV2
NAND2
B2
LDOUT
Q
D
CK
LVOUT
R
Q_{VDOUT}
LAOUT
Q_{AOUT}
INV1
NAND1
B1
LDIN
LVIN
Q_{VIN}
LAIN
Q_{AIN}
8
PH1
PH0
IN DATA(7:0)
NOTRESET0
IN_VALID
IN_ACCEPT

FIG. 4

FIG. 5(A)


00 aa 55 33 00 cc 00 04 00 aa 55 33 cc 00
00 aa 55 33 00 cc 00 04 00 aa 55 33 cc
00 aa 55 33 cc 00 04 aa 55 33 cc
110 150 190 230 270 310 350 390 430 470 510 550 590 630

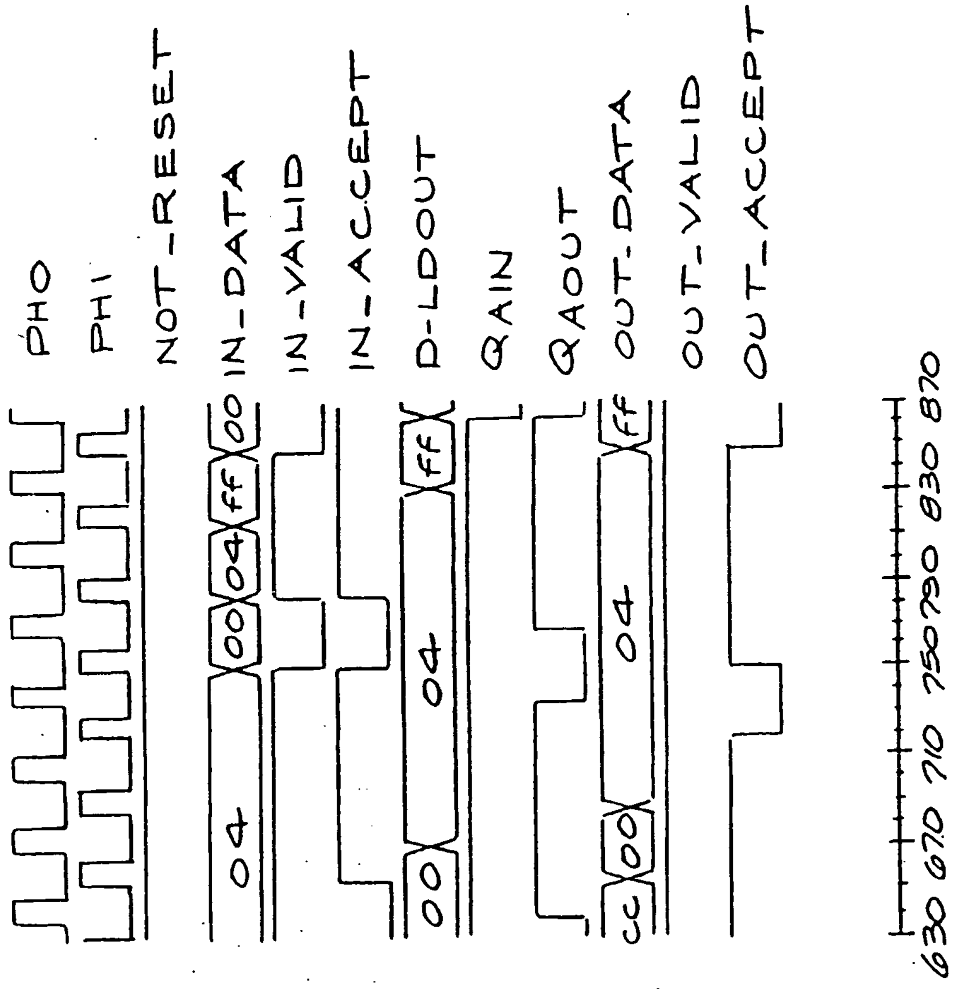

PHO
PHI
NOT_RESET
IN_DATA
IN_VALID
IN_ACCEPT
D-LDOUT
QAIN
QAOUT
OUT_DATA
OUT_VALID
OUT_ACCEPT
04
00
04
ff
00
00
04
ff
cc
00
04
ff
630
670
710
750
790
830
870

FIG. 5(B)

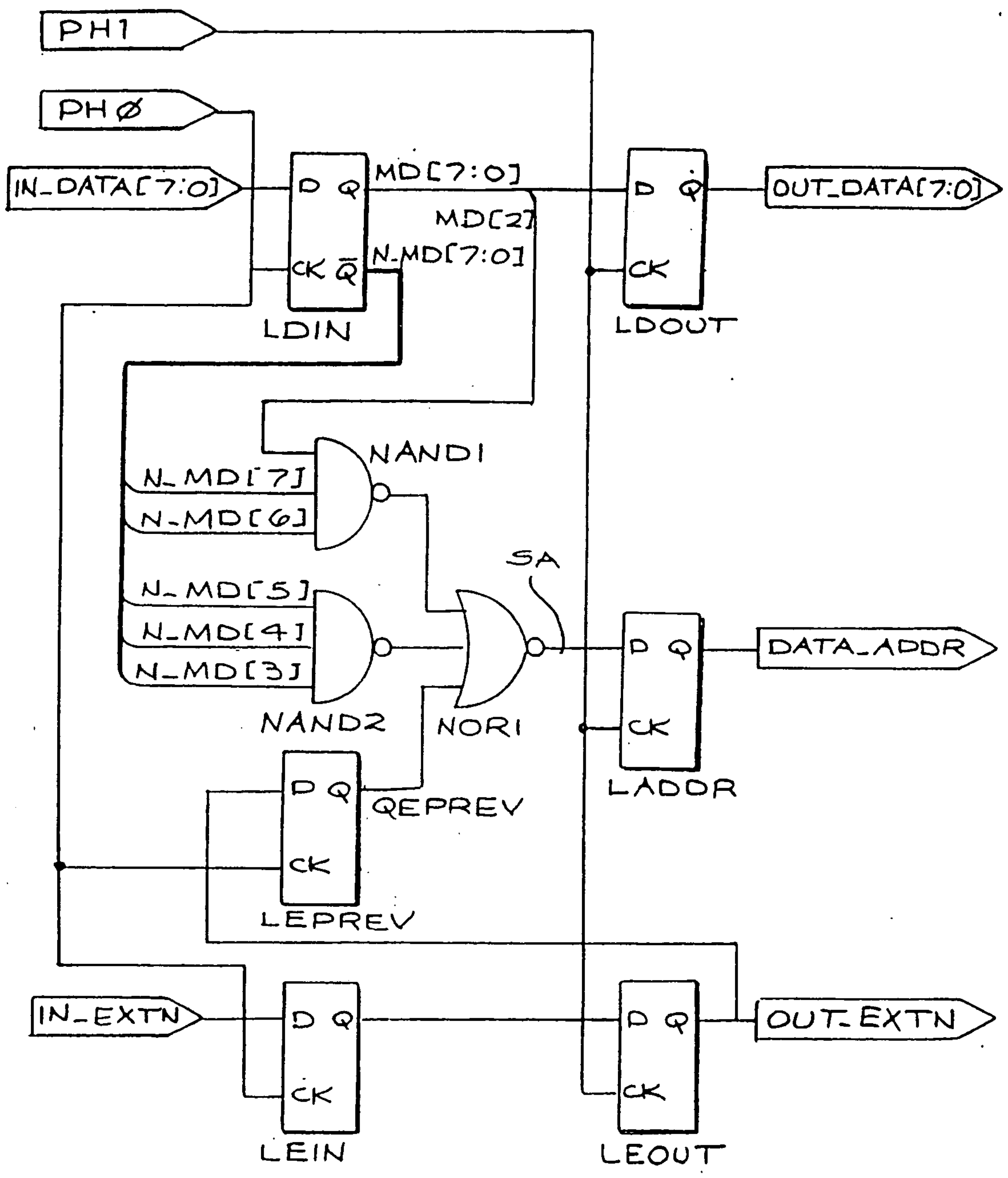

PH1
PHØ
IN_DATA[7:0]
D Q
CK Q̄
LDIN
MD[7:0]
MD[2]
N_MD[7:0]
LDOUT
OUT_DATA[7:0]
NAND1
N_MD[7]
N_MD[6]
SA
N_MD[5]
N_MD[4]
N_MD[3]
NAND2
NOR1
DATA_ADDR
LADDR
QEPREV
LEPREV
IN_EXTN
LEIN
LEOUT
OUT_EXTN

FIG. 6

FIG. 7

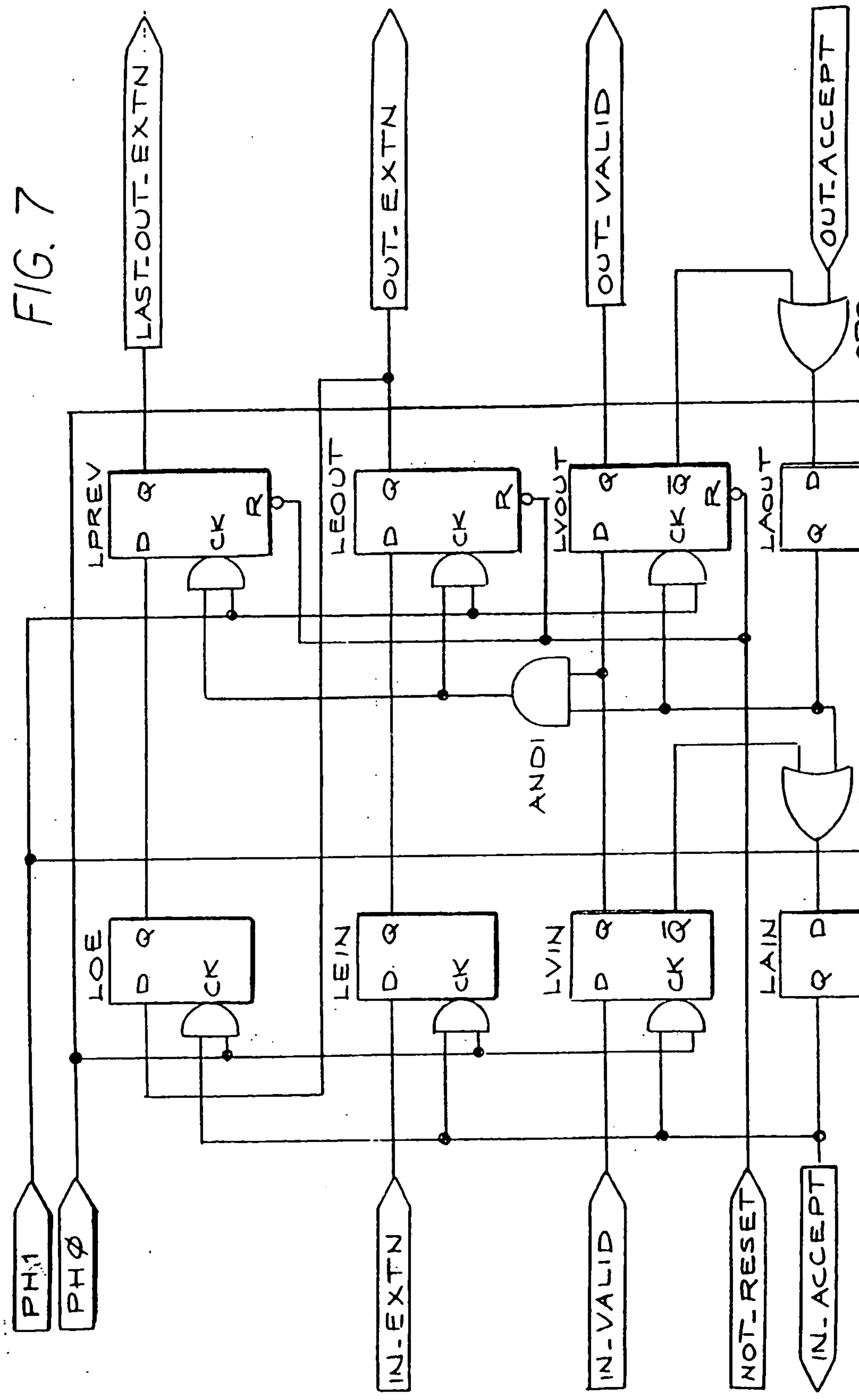

LAST-OUT-EXTN
OUT-EXTN
OUT-VALID
OUT-ACCEPT
OR2
LPREV
LEOUT
LVOUT
LAOUT
Q
D
CK
R
Q̄
AND1
OR1
LOE
LEIN
LVIN
LAIN
PH1
PHØ
IN-EXTN
IN-VALID
NOT-RESET
IN-ACCEPT

FIG. 8(A)

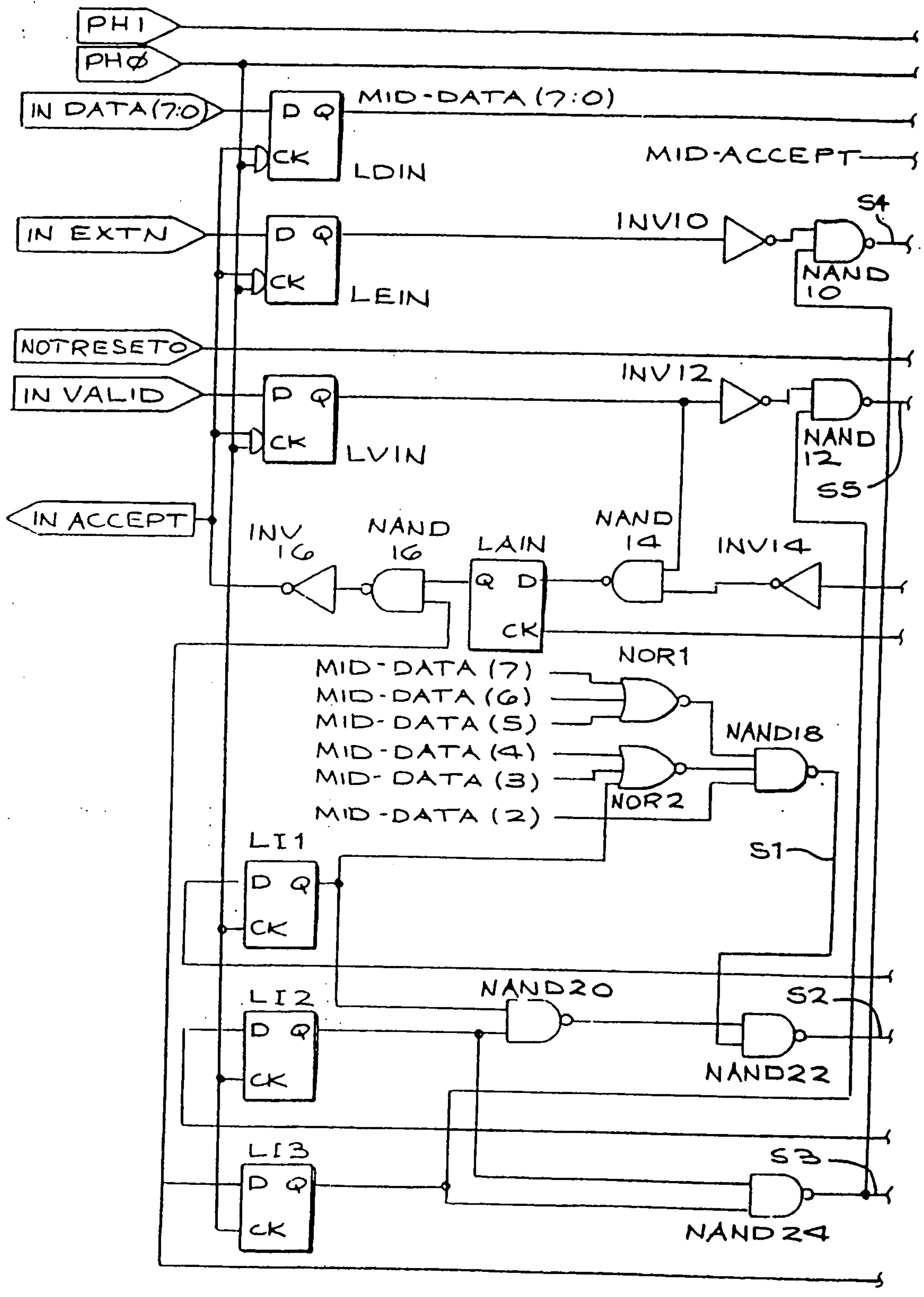

PH1
PHØ
IN DATA(7:0)
D Q
CK
MID-DATA(7:0)
LDIN
MID-ACCEPT
IN EXTN
LEIN
INV10
NAND 10
S4
NOTRESETO
IN VALID
LVIN
INV12
NAND 12
S5
IN ACCEPT
INV 16
NAND 16
LAIN
NAND 14
INV14
MID-DATA (7)
MID-DATA (6)
MID-DATA (5)
MID-DATA (4)
MID-DATA (3)
MID-DATA (2)
NOR1
NOR2
NAND18
S1
LI1
LI2
NAND20
NAND22
S2
LI3
S3
NAND24

FIG. 8(B)

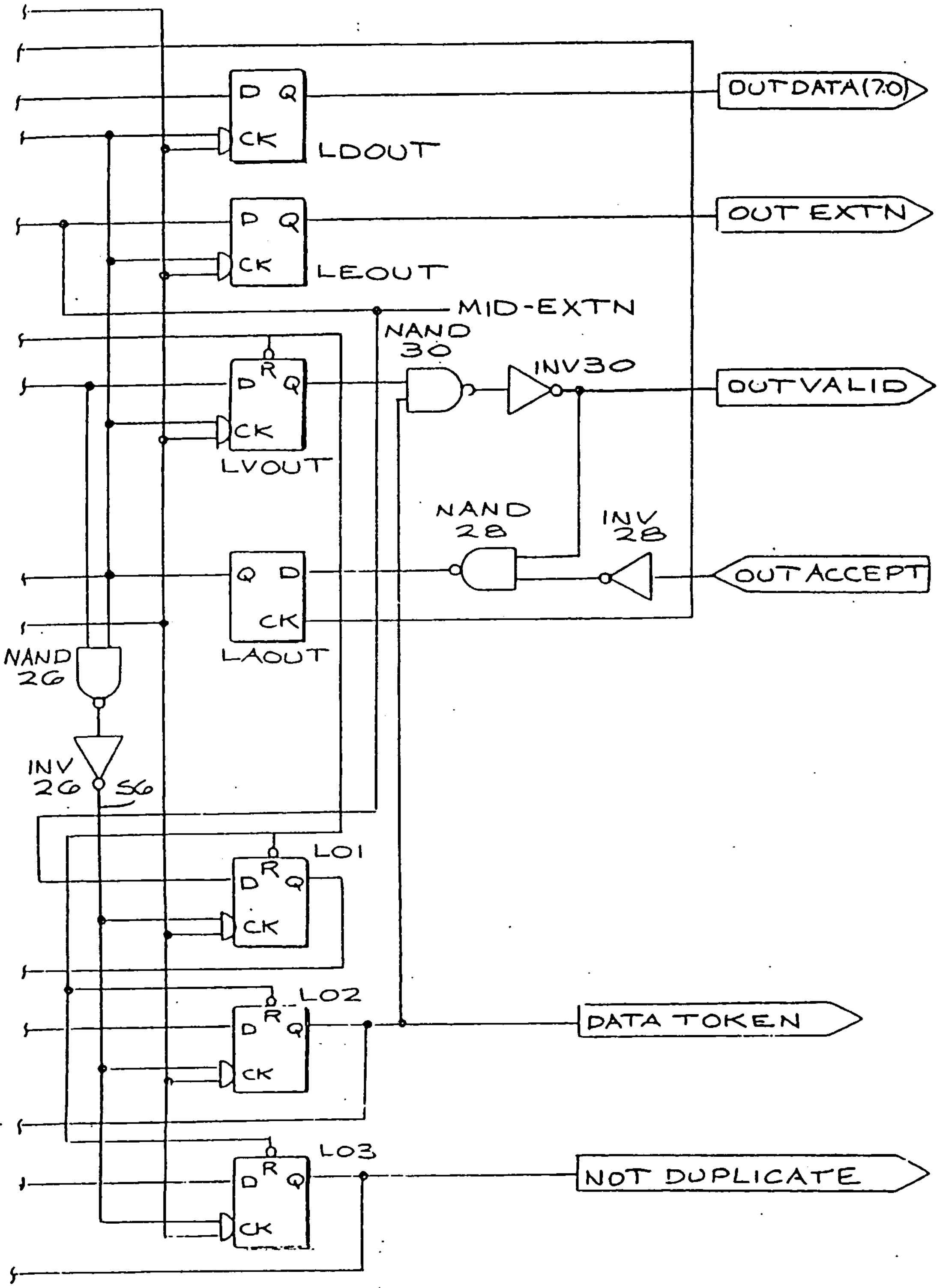
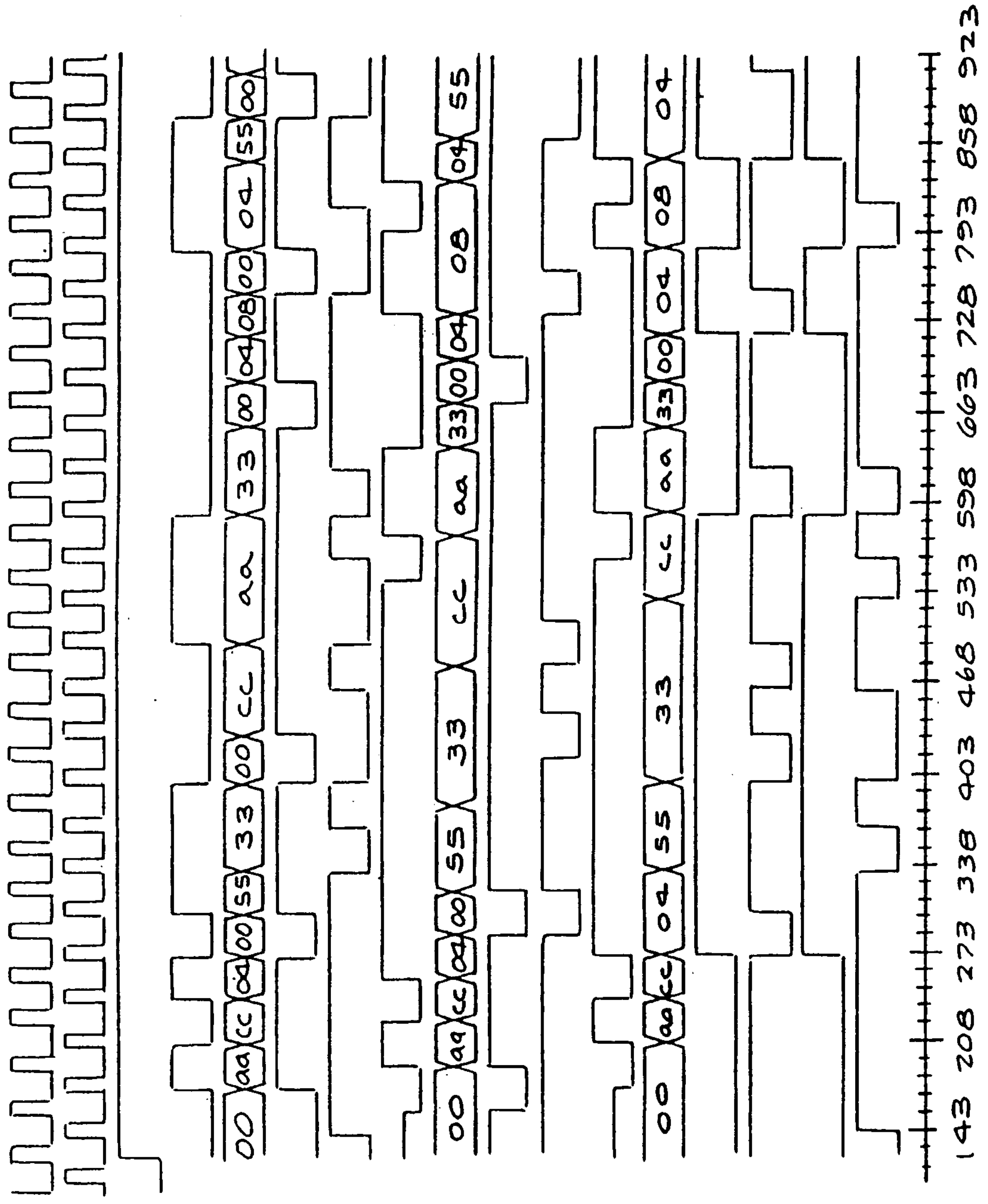

OUTDATA(7:0)
D Q
CK
LDOUT
D Q
CK
LEOUT
OUT EXTN
MID-EXTN
NAND 30
INV30
OUTVALID
D R Q
CK
LVOUT
NAND 28
INV 28
Q D
CK
LAOUT
OUT ACCEPT
NAND 26
INV 26
56
LO1
LO2
LO3
DATA TOKEN
NOT DUPLICATE

FIG. 9(A)


00 aa cc 04 00 55 33 00 cc aa 33 00 04 08 00 04 55 00
00 aa cc 04 00 55 33 cc aa 33 00 04 08 04 55
00 aa cc 04 55 33 cc aa 33 00 04 08 04
143 208 273 338 403 468 533 598 663 728 793 858 923

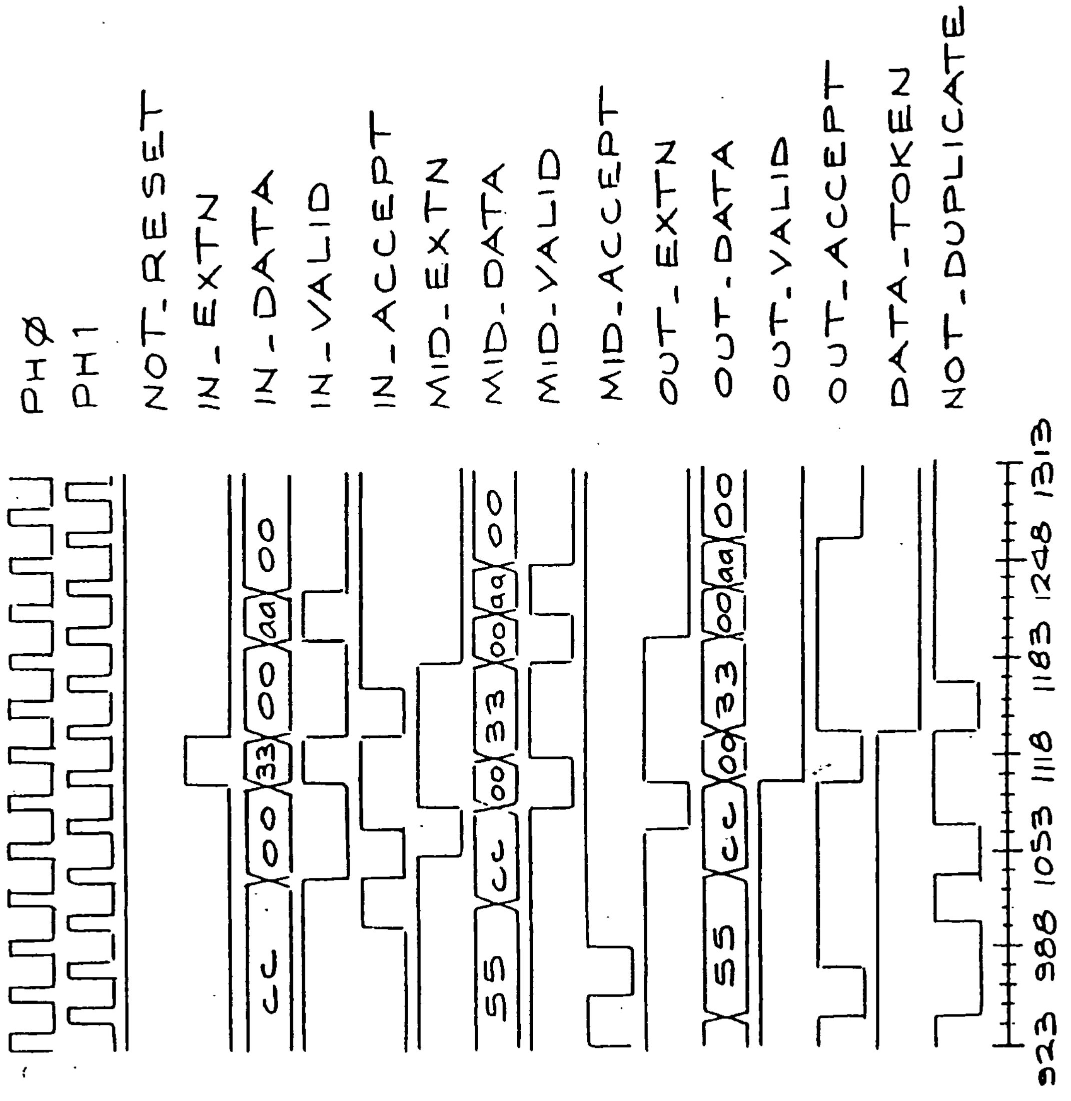

PHØ
PH1
NOT_RESET
IN_EXTN
IN_DATA
IN_VALID
IN_ACCEPT
MID_EXTN
MID_DATA
MID_VALID
MID_ACCEPT
OUT_EXTN
OUT_DATA
OUT_VALID
OUT_ACCEPT
DATA_TOKEN
NOT_DUPLICATE
cc
00
33
00
aa
00
55
cc
00
33
00
aa
00
55
cc
00
33
00
aa
00
923
988
1053
1118
1183
1248
1313

FIG. 9(B)

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 92 30 6038

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int. Cl.5)
X A	WESCON TECHNICAL PAPERS, no. 4/6, 30 October 1984, ANAHEIM, US pages 1 - 10 CHONG 'A data-flow architecture for digital image processing' * page 2, left column, lines 4-23; page 2, right column, line 35 - page 5, right column, line 16; page 8, left column, line 19 - right column, line 16; figures 3,5,6,9 *	16,20,23 1,18,22,27,31,36	G06F9/38
X A	US-A-4 789 927 (HANNAH) * the whole document *	20 1,18,22,30,31	
X A	IEEE JOURNAL OF SOLID-STATE CIRCUITS vol. 23, no. 1, February 1988, NEW YORK US pages 111 - 117 KOMORI ET AL. 'An elastic pipeline mechanism by self-timed circuits' * sections I-III; figures 1-5 *	16 1,18,23,27	TECHNICAL FIELDS SEARCHED (Int. Cl.5)
A	US-A-4 149 242 (PIRZ) * the whole document *	1,20,22,30,31	G06F
A	US-A-3 962 685 (BELLE ISLE) * column 2 , lines 1-40; column6, line 22 - column 7, line 53; column 8,lines 1-64 *	20,22	

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	25 FEBRUARY 1993	WEINBERG L.F.

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document